Tomita-tyylisistä yleistetyistä LR-jäsentäjistä

Jaakko Korpela

Tampereen yliopisto
Tietojenkäsittelytieteiden laitos
Tietojenkäsittelyoppi
Pro gradu -tutkielma
Marraskuu 2004

Tampereen yliopisto
Tietojenkäsittelytieteiden laitos
Tietojenkäsittelyoppi
Jaakko Korpela: Tomita-tyylisistä yleistetyistä LR-jäsentäjistä
Pro gradu -tutkielma, 92 sivua
Marraskuu 2004

Jäsentäjä on ohjelma, joka analysoi syötteenä annetun merkkijonon kieliopillisen rakenteen. Jäsentäjän tehtävänä on määrittää, kuinka kieliopista voidaan johtaa annettu syöte. Jos syötemerkkijono kuuluu kieliopin määräämään kieleen, jäsentäjä tulostaa sen jäsennyksen, muussa tapauksessa jäsentäjä ilmoittaa virheestä. LR-jäsentäminen on käytännössä suosittu jäsennysmenetelmä, jota on menestyksellisesti käytetty ohjelmointikielten kääntäjissä jo yli kolmenkymmenen vuoden ajan. Menetelmänä se on ilmaisuvoimainen, tehokas ja sopii automaattisille jäsentäjägeneraattoreille. LR-kieliopin määrittäminen halutulle kielelle voi kuitenkin olla hankalaa. Yleistetyn LR-jäsentämisen tarkoituksena on laajentaa LR-jäsentäjä toimimaan kaikilla kontekstittomilla kieliopeilla. Lang ratkaisi ensimmäisenä yleisen yleisen jäsennysongelman teoreettisesti ja Tomita kehitti myöhemmin tehokkaan käytännöllisen ratkaisun. Alkuperäinen Tomitan algoritmi ei kuitenkaan pysähdy käsitellessään kielioppia, joka sisältää epäsuoran vasemman rekursion. Tässä tutkielmassa perehdytään yleisesti Tomitan kehittämään yleistettyyn LR-jäsentämiseen. Työssä esitetään myös sekä Farshin että Scottin ja muiden jäsennysalgoritmit, jotka perustuvat Tomitan algoritmeihin, mutta toimivat oikein kaikilla kontekstittomilla kieliopeilla. Tutkielmassa analysoidaan myös alkuperäisessä Tomitan algoritmissa olevia ongelmia ja esitetään algoritmiin korjauksia perusteluineen.

Avainsanat ja -sanonnat: yleistetty kontekstiton jäsentäminen, Tomita-jäsentäjät.

Sisällys

1. Johdanto

Huomattava tulos determinististen jäsentäjäalgoritmien alalla neljäkymmentä vuotta sitten oli se, että kielet, jotka ovat lähes deterministisiä, voitiin tehokkaasti jäsentää käyttäen senaikaisia, tehokkuudeltaan suhteellisen pieniin koneisiin mahtuvia jäsentäjiä [JSE04]. Automaattisen jäsentämisen käytäntö onkin muuttunut suhteellisen vähän 1970-luvun alkupuolelta lähtien, jolloin julkaistiin useita LALR(1)- ja LL(1)-jäsentäjägeneraattoreita [MN04].

Suurin ongelma LALR(1)-jäsentäjägeneraattoreita käytettäessä on se, että sopivan kieliopin muodostaminen annetulle kieliopille on työlästä. LALR(1)-jäsentäjä on deterministinen, sillä jokaisella toiminta-askeleella on päätettävä käyttäen pinon pinnalla olevaa tilaa ja seuraavaa syötemerkkiä, siirretäänkö syötteestä seuraava syötemerkki pinoon vai valitaanko yksi kieliopin säännöistä sovellettavaksi redusoinniksi pinossa [ASU86]. Jäsennyskonfliktien ratkaiseminen vaatii algoritmin syvällistä ymmärtämistä, eikä se silloinkaan ole helppoa [MN04].

Kaikilla kontekstittomilla kielillä ei ole LALR(1)-kielioppia. Vaikka kielioppi voitaisiinkin muuttaa LALR(1)-kieliopiksi, prosessi voi olla työläs, ja se voi tuhota kieliopin käsitteellisen rakenteen. Sen sijaan että kuvattaisiin kieli, jota jäsennetään, päädytään kuvaamaan prosessi, jolla jäsennys tehdään. Koska kielioppi on tärkein osa jäsennysmäärittelyä, tämä ei ole toivottavaa. Lisäksi on huomattava, että silloin, kun kielen rakenne ei ole jäsentäjän kirjoittajan hallinnassa, ei voida olettaa kielen omaavan deterministä kielioppia. Luonnollisten kielten jäsentämisessä (natural language parsing, NLP) kielet ovat yleensä epädeterministisiä ja moniselitteisiä. Niinpä luonnollisen kielen jäsentäjän on löydettävä jäsennys mahdollisesti äärettömästä joukosta [JSE04].

Pääasiallinen vaihtoehto taistelussa siirrä/redosoi-konfliktien kanssa on ollut hylätä automaattinen jäsennysteknologia kokonaan. Kokemus on kuitenkin osoittanut, että jäsentäjän kirjoittaminen käsin on työlästä ja lopputulosta on vaikeata muuttaa sisältämään laajennuksia [MN04].

Toinen vaihtoehto jäsennyskonfliktien ratkaisemiseksi LR-jäsentämisessä on simuloida epädeterministisyyttä deterministisellä koneella, jolloin jäsennysohjelma suorittaa kaikki eteentulevat jäsennystoiminnot.

Tehokkaampien jäsentäjägeneraatorien kehitystä ja käyttöönottoa on osaltaan edistänyt valtava kehitys prosessoreiden tehokkuudessa sekä, ehkä tärkeämpänä asiana, käytettävän muistin määrässä [JSE04]. 1970-luvun alussa kun LALR- ja LL-jäsentäjägeneraatoreita kehitettiin, muistin koko oli

tyypillisesti 10^4 tavun luokkaa, kun tyypillisessä käyttökoneessa muistin määrä on nykyään 10^9 tavun luokkaa.

Tämän tutkielman sisältö on seuraava. Luvussa 2 esitellään aiheeseen liittyviä merkintöjä ja termejä. Luvussa 3 paneudutaan lyhyesti yleistettyjen LR-jäsentäjien kehitykseen esitellen sen merkittävimmät kehitysvaiheet. Luvussa 4 käsitellään normaalia LR-jäsentäjää. Luvussa 5 käsitellään yksityiskohtiin paneutumatta yleistetyn LR-jäsentämisen periaatteet esimerkkien kautta. Luvussa 6 annetaan formaali kuvaus luvun 5 menetelmistä ja todetaan, ettei se kaikissa tapauksissa pysty tunnistamaan syötteitä. Luvussa 6 esitetään myös lyhyesti esimerkin kautta Tomitan algoritmi kontekstittomille kieliopeille ja esitään siihen liittyvät ongelmat. Luvussa 7 esitellään Farshin jäsennysalgoritmi ja luvussa 8 Scottin ja muiden jäsennysalgoritmi. Luvussa 9 palataan Tomitan kontekstittomien kielioppien jäsennysalgoritmiin ja annetaan sen formaali kuvaus. Samassa luvussa analysoidaan myös tarkemmin algoritmiin liittyviä ongelmia. Luvussa 10 tuodaan esille korjauksia Tomitan algoritmiin ja luvussa 11 esitellään joitakin korjauksiin liittyviä todistuksia. Vastaavanlaisia korjauksia alkuperäiseen Tomitan algoritmiin ei kirjallisuudessa ole aikaisemmin esitetty. Kaikki tutkielmassa esitetyt tulokset ovat kirjoittajan omaa käsialaa.

2. Merkintöjä ja termejä

Kontekstiton kielioppi koostuu joukosta **T** perusmerkkejä, joukosta **N** apumerkkejä, merkistä $S \in \mathbf{N}$, jota sanotaan alkumerkiksi, ja joukosta kieliopin sääntöjä. Säännöt ovat muotoa $A ::= \alpha$, missä $A \in \mathbf{N}$ ja α on (mahdollisesti tyhjä) perus- ja apumerkeistä koostuva merkkijono.

Jäsennysaskel (derivation step) on muotoa $\gamma A \sigma \Rightarrow \gamma\alpha\sigma$ oleva toiminta, missä γ ja σ ovat perus- ja apumerkeistä koostuvia merkkijonoja ja $A ::= \alpha$ on kieliopin sääntö. Merkkijonon τ jäsennys merkkijonosta σ on sarja jäsennysaskelia $\sigma \Rightarrow \beta_1 \Rightarrow \beta_2 \Rightarrow ... \Rightarrow \beta_{n-1} \Rightarrow \tau$. Jos τ jäsentyy merkkijonosta σ käyttämällä *n* jäsennysaskelta, voidaan kirjoittaa myös $\sigma \overset{*}{\Rightarrow} \tau$ tai $\sigma \overset{n}{\Rightarrow} \tau$.

Jos jäsennyksiä tehtäessä korvataan aina vasemmanpuoleisin apumerkki, puhutaan *vasemmanpuoleisimmasta jäsennyksestä* (leftmost derivation) tai *vasemmasta johdosta*. Vastaavasti korvattaessa aina oikeanpuoleisin apumerkki puhutaan *oikeanpuoleisimmasta jäsennyksestä* (tai *oikeasta johdosta*). Vasemmanpuoleisinta jäsennystä merkitään $\alpha \underset{vp}{\Rightarrow} \beta$ ja oikeanpuoleisinta jäsennystä merkitään $\alpha \underset{op}{\Rightarrow} \beta$.

Tyhjästä merkkijonosta käytetään merkintää ε. Kieliopin säännön $A ::= \alpha\beta$ sanotaan olevan *oikealta tyhjentyvä* (right nullable), jos $\beta \neq \varepsilon$, mutta $\beta \overset{*}{\Rightarrow} \varepsilon$. Kieliopin sanotaan omaavan *epäsuoran vasemman (tai oikean) rekursion* (hidden left/right recursion), jos siinä on apumerkki *A* ja sellaiset merkkijonot τ ja σ, että $\tau \overset{+}{\Rightarrow} \varepsilon$ ja $A \overset{*}{\Rightarrow} \sigma A \tau$ (tai $A \overset{*}{\Rightarrow} \tau A \sigma$).

Kieliopillinen muoto (sentential form) on sellainen merkkijono α, että $S \overset{*}{\Rightarrow} \alpha$ ja *lause* on sellainen kieliopillinen muoto, joka koostuu vain perusmerkeistä. Kieliopin Γ muodostama *kieli* määritellään niistä lauseista koostuvaksi joukoksi $L(\Gamma)$, joilla on jäsennys kieliopin Γ alkumerkistä *S*.

Lausetta, joka on generoitu kieliopista käyttäen vasemmanpuoleisinta jäsennystä, sanotaan *vasemmanpuoleiseksi kieliopilliseksi muodoksi*. Oikeanpuoleisen jäsennykseksen generoimaa lausetta sanotaan vastaavasti *oikeanpuoleiseksi kieliopilliseksi muodoksi*. Alkuosan $\alpha\beta$ oikeanpuoleisesta kieliopillisesta muodosta $\alpha\beta\omega$ sanotaan olevan kieliopin *kelvollinen alkuosa* (viable prefix).

Kieliopin symbolille *x* ja merkkijonolle γ määritellään

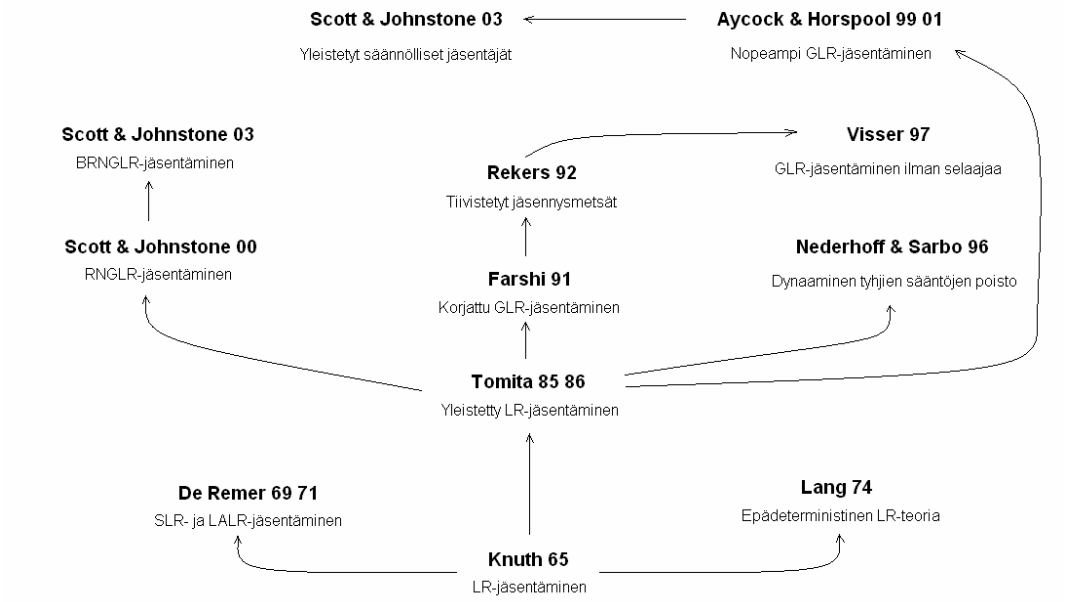
$$FIRST_T(x) = \{t \in \mathbf{T} \mid \text{jollekin merkkijonolle } \beta,\ x \overset{*}{\Rightarrow} t\beta\},\ FIRST_T(\varepsilon) = \{\varepsilon\}$$

$$FIRST(x\gamma) = \begin{cases} FIRST_T(x) \cup FIRST_T(\gamma), \; jos \;\; x \overset{*}{\Rightarrow} \varepsilon \\ FIRST_T(x), \;\; muuten. \end{cases}$$

Äärellinen tila-automaatti (finite state automaton, FA) on suunnattu graafi, jonka solmuja sanotaan tiloiksi. Yksi tila automaatissa on *alkutila* ja jotkut tilat ovat *hyväksymistiloja* tai *lopputiloja*. Graafin kaaria kutsutaan *siirtymiksi* ja niitä merkitään syötesymboleilla tai tyhjällä merkkijonolla ε. Äärellisiä automaatteja käytetään syötemerkkijonojen hyväksymiseen tai hylkäämiseen. Äärellinen automaatti hyväksyy syötteen, jos alkutilasta lähtien kulkemalla syötemerkkijonon määräämien symboleiden osoittamia siirtymiä päädytään lopputilaan, kun koko syötemerkkijono on luettu. Muussa tapauksessa automaatti hylkää syötteen. Äärellinen tila-automaatti on *deterministinen* (deterministic finite automata, DFA), jos siinä ei ole tyhjiä siirtymiä, eli siirtymiä, jossa syötesymbolia ei lueta, ja jos jokaiselle kieliopin symbolille x jokaisesta solmusta on korkeintaan yksi siirtymä symbolilla x.

3. Yleistettyjen LR-jäsentäjien kehitys

Backus-Naurin notaation (Backus-Naur Form, BNF) kehittäminen ohjelmointikieli ALGOL-60 syntaksin kuvaamiseksi [BBG+63] oli ensimmäinen merkittävä askel jäsennysmenetelmien formalisoimiseksi. BNF:n avulla mukaan saatiin liitettyä Chomskyn kehittämä kontekstittomien kielioppien matemaattinen viitekehys [GR62]. Kuvassa 3.1 on hahmoteltu yleistetetyn LR-jäsentämisen kehitystä. Knuth [Knu65] formalisoi kokoavan LR-jäsennysmenetelmän ja osoitti, että LR-jäsentäjät toimivat lineaarisessa ajassa syötteen pituuden suhteen. Knuthin työ sisälsi suurimman osan teoreettisista tuloksista LR-jäsentämisestä, mutta Knuth ei itse pitänyt tuloksiaan käytännössä sovellettavina. Vaikka Knuthin kuvailemat LR-jäsentäjät toimivat lineaarisessa ajassa syötteeseen nähden, niiden tilavaatimus on pahimmassa tapauksessa eksponentiaalinen kieliopin kokoon nähden. LR-jäsentäjän vaatima muisti käytännössä käytettävillä kielillä (puhumattakaan teoreettisista ääritapauksista) ylitti selvästi senaikaisten koneiden kapasiteetin.



Kuva 3.1. Yleistetyn LR-jäsentämisen kehitys [JSE04].

Tilanne muuttui De Remerin LALR- [DeR69] ja SLR-algoritmien [DeR71] ansiosta. Nämä algoritmit loivat pohjan Knuthin tulosten käytännölliseen

soveltamiseen. De Remerin algoritmit käyttävät samaa jäsennysmallia kuin Knuthin algoritmi, mutta jäsennyksessä käytettävien jäsennystaulujen rakentamiseen käytetty menetelmä tuottaa pienempiä jäsennystauluja, säilyttäen kuitenkin suuren osan Knuthin mallin ilmaisuvoimasta. Unix-työkalu YACC [Joh75] käyttää LALR-jäsentämisalgoritmia ja joitakin menetelmiä [AJU75] kieliopin moniselitteisyyden purkamiseksi. YACC muodostui nopeasti standardi-työkaluksi jäsentämisessä.

Akateemisemmalla puolella Lang [Lan74] työskenteli teoreettisten kysymysten parissa, tutkien mahdollisuuksia simuloida epädeterministisyyttä deterministisillä menetelmillä.

Tomita [Tom86] kehitti yleistetyn LR-jäsentäjän (Generalized LR, GLR) luonnollisten kielten tehokkaaseen jäsentämiseen. Tomita huomioi, että luonnollisten kielten kieliopit ovat enimmäkseen LR-kielioppeja, joissa on jonkin verran moniselitteisyyttä. Samaa voidaan sanoa monista ohjelmointikielistä (varsinkin esim. C++), joten Tomitan algoritmi soveltuu hyvin myös niiden jäsentämiseen. Vaikka ei ole varmaa, oliko Tomita tietoinen Langin tuloksista, Tomitan työtä voidaan pitää näiden käytännöllisenä toteutuksena, jossa johdon tallentamiseen käytetty tietorakenne on pääasiallinen uusi tulos.

GLR käyttää syvyys-ensin –etsintää ylläpitäen listaa kaikista mahdollisista (osittaisista) jäsennyksistä nykyiseen syötemerkkiin asti. LR-jäsentäjässä konfiguraatio on tallennettuna tilanumeroin pinoon. Niinpä tietyn syötteen kaikkien jäsennysvaihtoehtojen läpikäymiseksi tarvitsee ylläpitää useampaa pinoa, yksi jokaista yksittäistä jäsennystä kohti. LR-jäsennysalgoritmia on muokattu niin, että aktiivisista pinoista (epäaktiiviset pinot ovat sellaisia, joissa on päädytty tilanteeseen, jossa jäsennystä ei voida jatkaa) suoritetaan kaikki mahdolliset redusoinnit, ennen seuraavan syötemerkin siirtämistä pinoon. Käytettäessä osoitinpohjaista esitystapaa pinoissa päädytään puurakenteeseen. Tomita havaitsi, että niiden arvojen joukko, joka voi esiintyä pinon pinnalla, on äärellinen joukko jäsennystiloja. Pinon toiminnan kontekstiton luonne johtaa siihen, että puun oksia voidaan yhdistää, kun niillä on sama lehden arvo [JSE04]. Tomita kutsui tietorakennetta graafirakenteiseksi pinoksi (Graph Structured Stack, GSS).

Tomita kuvasi viisi algoritmia. Näistä ensimmäinen kuvaa aiheeseen liittyviä perusideoita ja soveltuu vain LR(1)-kieliopeille. Toinen algoritmi toimii säilyttävillä kieliopeilla ja lopuissa algoritmeissa lisätään ominaisuuksia tyhjien sääntöjen hallitsemiseksi. Tomitan algoritmeista löytyy kuitenkin virhe, sillä ne eivät pysähdy kieliopeilla, jotka sisältävät epäsuoran vasemman rekursion.

Farshin [NF91] ratkaisu ongelmaan oli suorittaa täydellinen etsintä kaikille aktiivisille tiloille uusien tilojen etsimiseksi. Tämä etsintä voi kuitenkin olla

epäkäytännöllisen paljon aikaa vievä. Rekers [Rek92] kehitti Farshin algoritmia tuottamalla tiiviimmän jäsennysmetsän. Visser [Vis97] taas muokkasi Farshin algoritmia toimimaan ilman selaajaa. Molemmissa algoritmeissa redusointien etsintä tyhjiin sääntöihin liittyen on kuitenkin jäljellä. Scottin ja muiden [SJH00] kehittämämä RNGLR-algoritmi sisältää hienovaraisemman korjauksen Tomitan algoritmeissa olevaan ongelmaan tyhjiin sääntöihin liittyen ja BRNGLR [SJE03] käyttää redusointien suorittamiseen ”binärisöintiä”, jolloin jäsentäjä toimii kuutiollisessa ajassa syötteeseen nähden mille tahansa kontekstittomalle kieliopille. Aycockin ja Horspoolin [AHJ01] jäsentäjät eivät suoranaisesti liity GLR-jäsentämiseen vaan ovat lähinnä saaneet joitakin vaikutteita tästä suunnasta.

4. LR-jäsentäjä

4.1. Johdanto

LR-jäsentäjä on kontekstittomien kielioppien jäsentäjätyyppi, jota yleisesti käytetään ohjelmointikielten kääntäjissä. LR-jäsentäjä on *kokoava* (bottom-up) jäsentäjä, joka lukee syötettään vasemmalta oikealle ja pyrkii käänteistä oikeaa johtoa muodostaen kohti kieliopin alkumerkkiä. LR-jäsentämisellä on lukuisia etuja ja se on pääasiallinen jäsennysmenetelmä monille ohjelmointikielille. Kaikista syötettä vasemmalta oikealle lukevista jäsentäjistä LR-jäsentäjä havaitsee syntaktiset virheet aikaisimmassa mahdollisessa vaiheessa.

LR-jäsentäjä koostuu *syötepuskurista,* josta syötettä luetaan merkki kerrallaan, *lukupäästä*, joka lukee aina jotain kohtaa syötteestä, *pinosta* ja *jäsennystaulusta*. Jäsennystaulussa on toiminnot jokaiselle perusmerkille jokaisessa tilassa ja siirry-tiloja (goto states) apumerkeille. Edellistä osaa taulusta sanotaan toimintatauluksi ja jälkimmäistä siirry-tauluksi. Pino koostuu symboli/tila pareista ja aluksi pinossa on vain tila 0.

LR-jäsentäjän tilat koostuvat *LR-alkioista* (LR-items). LR-alkio on pari $([A ::= \alpha \bullet \beta],\ \omega)$, missä ω on perusmerkkijono, joka voi koostua perus-merkeistä ja syötteen loppua kuvaavasta $-merkistä. LR-alkion osaa $[A ::= \alpha \bullet \beta]$ sanotaan *ytimeksi* (core). Pari $([A ::= \alpha \bullet \beta],\ \omega)$ kuvaa jäsentäjän ”kontekstin”. Tällöin yritetään löytää merkkijono, joka jäsentyy kieliopissa apumerkistä A, ja jota seuraa merkkijono ω. Merkkijono α on jo pinon pinnalla ja tämän jälkeen jäljellä olevasta syötteestä on löydettävä osa, joka jäsentyy merkkijonosta $\beta\omega$. Jos ω on aina vain yhden merkin mittainen, sanotaan sitä *kurkistusalkioksi* (lookahead item). Tällöin LR-jäsentäjää sanotaan LR(1)-jäsentäjäksi. Kurkistusalkiolla ei ole merkitystä muotoa $[A ::= \alpha \bullet \beta, a]$, missä $\beta \neq \varepsilon$, olevissa LR(1)-alkioissa, mutta tilassa, jossa on muotoa $[A ::= \alpha \bullet, a]$ oleva LR(1)-alkio, suoritetaan *redusointi*, jos seuraava syötemerkki on a.

Toimintataulu voi sisältää neljä erilaista toimintoa jokaiselle tilalle:

- Syötemerkin siirto pinoon ja siirtyminen tilaan s
- Redusointi säännön p mukaan
- Virhe
- Syötemerkkijonon hyväksyminen.

Algoritmi etenee lukemalla nykyisen tilan pinon pinnalta ja seuraavan syötemerkin syötepuskurista. Näiden perusteella on toimintataulussa määrätty toiminta. Syötemerkin siirrossa pinoon työnnetään ensimmäinen merkki syötepuskurista ja uusi tila. Redusoinnissa pinosta poistetaan sopiva määrä symboleita ja katsotaan pinon pinnalle jäävä tilasymboli *t*. Tämän jälkeen työnnetään pinoon säännön p vasemman puolen apumerkki *A* sekä se tila, joka on siirry-taulun kohdassa (*t*, *A*). Virhetoiminnossa siirrytään virheestä-toipumustilaan ja hyväksymistoiminnossa syöte hyväksytään.

Redusoinnissa pinosta poistettavien symbolien määrä määräytyy redusoinnissa käytettävän säännön mukaan. Jos säännön oikeassa puolessa on m merkkiä, poistetaan pinosta $2m$ symbolia (m tilaa ja m merkkiä).

4.2. LR(1)-jäsentäjän muodostaminen

LR(1)-alkio $[A ::= \alpha \bullet \beta, a]$ on *pätevä* (valid) kelvolliselle alkuosalle γ, jos kieliopissa on jäsennys $S \underset{op}{\overset{*}{\Rightarrow}} \delta A w \underset{op}{\Rightarrow} \delta\alpha\beta\, w$, missä

1. $\gamma = \delta\alpha$, ja
2. joko a on merkkijonon w ensimmäinen symboli, tai $w = \varepsilon$ ja $a = \$$ [ASU86].

Yleisesti ottaen kelvolliselle alkuosalle voi olla useita päteviä alkioita.

LR(1)-jäsentäjän muodostamisessa määrätään pätevät alkiot jokaiselle kelvolliselle alkuosalle kieliopissa (alkaen tyhjästä merkkijonosta). Pätevien alkioiden joukosta kelvolliselle alkuosalle γ käytetään merkintää V(γ). Jokainen V(γ) vastaa yhtä jäsentäjän tilaa.

Keskeinen ajatus LR(1)-jäsentäjän rakentamisessa on ensin muodostaa kieliopista detarministinen äärellinen automaatti, joka tunnistaa kelvolliset alkuosat. LR(1)-alkiota ryhmitellään joukkoihin *I*, jotka muodostavat yllä mainitut LR(1)-jäsentäjän tilat. Tilojen muodostamiseksi määritellään kaksi funktiota, *sulkeuma* ja *siirry*.

Sulkeuma-funktiossa tilassa *s*, jossa on muotoa $[A ::= \alpha \bullet B\beta, a]$ olevia LR(1)-alkioita, lisätään tilaan *s* myös alkiot $[B ::= \bullet\gamma, b]$, missä $B ::= \gamma$ on kieliopin sääntö ja *b* = FIRST(βa) ensimmäinen perusmerkki lauseessa, joka on jäsennetty merkkijonosta βa. Perusteluna tälle on se, että LR(1)-alkion $[A ::= \alpha \bullet B\beta, a]$ mukaan oltaessa tilassa *s*, pinon pinnalla on merkkijono α ja seuraavaksi odotetaan syötteestä löytyvän merkkijono, joka on jäsennettävissä merkkijonosta $B\beta a$. Koska *B* on säännön $B ::= \gamma$ vasen puoli, seuraa tästä, että

tilassa s odotetaan seuraavaksi syötteestä löytyvän merkkijonon, joka on jäsennettävissä merkkijonosta γb.

Siirry-funktio saa syötteenään parin (I, X), missä I on LR(1)-alkiojoukko ja X on kieliopin symboli. Joukko *siirry*(I, X) määritellään muotoa [$A ::= \alpha X \bullet \beta, a$] olevien alkioiden sulkeumaksi, kun joukossa I on alkio [$A ::= \alpha \bullet X\beta, a$]. Intuitiivisesti, jos I on LR(1)-alkiojoukko, joka on pätevä kelvolliselle alkuosalle γ, niin siirry(I, X) on LR(1)-alkiojoukko, joka on pätevä kelvolliselle alkuosalle γX.

4.3. Alkiojoukkojen muodostaminen LR(1)-jäsentäjälle

Alkiojoukkojen I muodostamiseksi laajennetaan aluksi kielioppi G kieliopiksi G' lisäämällä sääntö $S' ::= S$, missä S on kieliopin alkumerkki ja S' on apumerkki, joka ei esiinny kieliopissa G. Tämän apumerkin tarkoituksena on tunnistaa kohta, jossa syöte voidaan hyväksyä. LR(1)-alkiojoukot muodostetaan algoritmilla 1.

Algoritmi 1. Algoritmi LR(1)-alkiojoukkojen muodostumiseen [ASU86].

Syöte. Laajennettu kielioppi G'.

Tulos. LR(1)-alkiojoukot, jotka ovat päteviä alkioita yhdelle tai useammalle kieliopin G' kelvolliselle alkuosalle.

Menetelmä (pääfunktio *alkiot*, sekä funktiot *sulkeuma* ja *siirry*):

funktio *sulkeuma(I)* ;
aloita
  toista
    jokaiselle alkiolle [$A ::= \alpha \bullet B\beta, a$] joukossa I,
          jokaiselle säännölle $B ::= \gamma$ kieliopissa G'
          ja jokaiselle perusmerkille $b =$ FIRST(βa)
          joilla [$B ::= \bullet\gamma, b$] ei ole joukossa I **suorita**
      lisää [$B ::= \bullet\gamma, b$] joukoon I
  kunnes joukkoon I ei voida enää lisätä alkioita
loppu;

funktio *siirry(I, X)*;
aloita
  olkoon J sellainen joukko LR(1)-alkioita [$A ::= \alpha X \bullet \beta, a$], että
      [$A ::= \alpha \bullet X\beta, a$] on joukossa I;
  palauta *sulkeuma*(J)
loppu;

funktio *alkiot*(*G'*)
aloita
  C = {*sulkeuma*({[*S*'::= •*S*,$]})}
  toista
    jokaiselle sellaiselle LR(1)-alkiojoukolle *I* joukossa *C* ja jokaiselle sellaiselle kieliopin ymbolille *X* , että *siirry(I, X)* ei ole tyhjä joukko, eikä se ole joukossa *C* **suorita**
      lisää *siirry(I, X)* joukkoon *C*
  kunnes joukkoon *C* ei voida enää lisätä uusi alkiojoukkoja *I*
loppu;

LR(1)-alkiojoukot muodostavat deterministisen äärellisen automaatin, joka tunnistaa kieliopin kelvolliset alkuosat. Automaatissa LR(1)-alkiojoukosta *I* on siirtymä kieliopin symbolilla *X* LR(1)-alkiojoukkoon *J*, jos *J* = siirry(*I*, *X*). Jokainen alkiojoukko *I* automaatissa on joukko päteviä LR(1)-alkioita sille kelvolliselle alkuosalle, joka muodostuu kuljetuista siirtymistä kuljettaessa alkutilasta tilaan *I*.

4.4. LR(1)-jäsennystaulun muodostaminen

LR(1)-jäsennystaulua käytetään jäsennyksessä jäsennystoiminnasta päätettäessä. Jäsennystaulun rivit koostuvat automaatin tiloista ja sarakkeet (laajentamattoman) kieliopin symboleista. Jäsennystaulu rakennetaan algoritmia 2 käyttäen.

Algoritmi 2. Algoritmi LR(1)-jäsennystaulun muodostamiseen [ASU86].

Syöte. Laajennettu kielioppi *G'*.

Tulos. LR(1)-jäsennystaulu.

Menetelmä:

1. Muodosta $C = \{ I_0, I_1, ..., I_n \}$, kokoelma joukoista LR(1)-alkioita kieliopille *G'*.

2. Jäsentäjän tila *i* muodostetaan joukosta I_i . Jäsennystoiminto tilassa *i* määräytyy seuraavasti:

   a) Jos LR(1)-alkio [$A ::= \alpha \bullet a\beta, b$] on joukossa I_j ,
      aseta *toiminta*[*i*, *a*] = ”siirrä *j*”. Tässä tapauksessa *a* on perusmerkki

b) Jos [$A ::= \alpha\bullet, a$] on joukossa I_i ja $A \neq S'$,
aseta *toiminta*[i, A] = ”redusoi $A ::= \alpha$ ”

c) Jos [$S' ::= S\bullet, \$$] on joukossa I_i, aseta *toiminta*[i, $\$$] = ”hyväksy”

Jos edellämainitut säännöt tuottavat konflikteja, eli useamman kuin yhden toiminnon yhteen toimintataulun kohtaan, kielioppi ei ole LR(1)-kielioppi ja algoritmin suoritus epäonnistuu.

3. Jos *siirry*(I_i, A) = I_j, niin jäsennystaulussa *siirry*[i, A] = j.

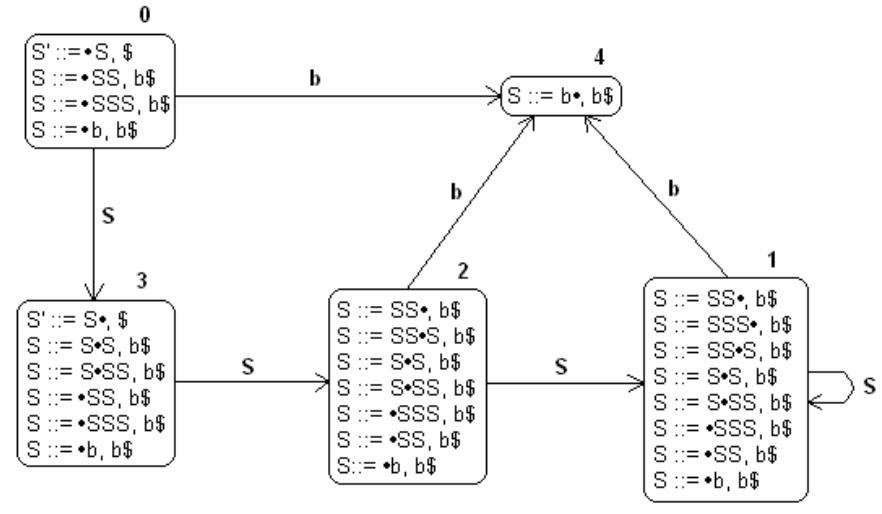
4. Kaikki kohdat jäsennystaulussa, johon kohdat 2 ja 3 eivät tuota merkintää, ovat virhetiloja.

5. Jäsentäjän alkutila on se tila, joka muodostetaan sulkeumalla LR(1)-alkiosta [$S' ::= \bullet S, \$$].

Esimerkki 1. Tarkastellaan esimerkkinä kielioppia Γ_1, jonka säännöt ovat

$S ::= SS$
$S ::= SSS$
$S ::= b.$

Tästä muodostettu LR(1)-alkiojoukkojen mukainen DFA on esitetty kuvassa 4.1.



Kuva 4.1. LR(1)-jäsennyksen mukainen DFA kieliopille Γ_1.

Tila 3 sisältää LR(1)-alkion $[S' ::= S\bullet, \$]$, joten tila 3 on lopputila. LR(1)-alkiojoukoista muodostettu LR(1)-jäsennystaulu on esitetty kuvassa 4.2.

	$	b	S
0		s4	g3
1	r1/s2	r1/r2/s4	g1
2	r1	r1/s4	g1
3	hyv.	s4	g2
4	r3	r3	

Kuva 4.2. LR(1)-jäsennystaulu kieliopille Γ_1.

5. GLR-jäsennysalgoritmista yleisesti

Aivan kuten LR-jäsentäjämisessä, GLR-jäsentämisessäkin käytetään jäsennyspinoa ja äärellistä ohjausta. Äärellinen ohjaus määrittelee, mikä jäsennystoiminto (syötemerkin siirto pinoon, redusointi) suoritetaan seuraavan syötemerkin ja pinon pinnalla olevan tilan perusteella. Mutta, kuten jo yllä mainittiin, pinon sijasta GLR-jäsentäjä käyttää graafirakenteista pinoa (GSS), joka sisältää kaikki pinot, jotka LR-jäsentäjä mahdollistaa. Jokaista pinoa graafissa pidetään erillisenä LR-jäsentäjänä ja jokaista pinoa käsitellään rinnakkain. Pinot synkronoidaan siirtämällä jokainen syötemerkki samanaikaisesti.

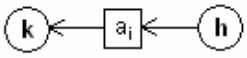
GLR-jäsentäjän graafirakenteinen pino on yksinkertainen. Jotkin pinojen solmuista ovat pinojensa pinnalla ja algoritmi pitää näistä lukua. Jokaisella solmulla on vähintään yksi sellainen suunnattu kaari alempiin solmuihin jossain pinossa, että jokainen äärellinen polku tietyn pinon pintasolmusta yksikäsitteiseen pohjasolmuun muodostaa potentiaalisen LR-jäsennyspinon. Kontekstittomissa kieliopeissa voi olla muotoa $A ::= \varepsilon$ olevia sääntöjä, ns. *tyhjiä sääntöjä*. Tyhjiä sääntöjä sisältävän kieliopin tapauksessa GSS voi sisältää syklin ja näin ollen äärettömän pitkän polun. Myöhemmin esitetään algoritmeja, joissa tämä on otettu huomioon. Nämä algoritmit toimivat oikein myös tyhjiä sääntöjä sisältävien kielioppien kanssa.

GLR toimii seuraavasti: jokaisen syötemerkin kohdalla jokaisen pinon pinnalla olevalle tilalle suoritetaan jokainen mahdollinen LR(1)-jäsennystoiminto. Toimintoja voi olla useita mahdollisia vastaten jäsennyskonfliktia tavallisessa LR-jäsentäjässä. Syötemerkin siirto pinoon luo uuden solmun pinon pinnalle ja tästä luodaan suunnattu kaari alempaan solmuun. Redusoinnissakin on mahdollista luoda uusi solmu, mutta toisin kuin LR-jäsentäjässä, jossa pinosta poistetaan tiloja ja laitetaan uusi tila pinoon, GLR:ssä kuljetaan GSS:ää kaikkien LR-jäsentäjässä poistettavien tilojen ohi. Tämän jälkeen luodaan uusi solmu (ellei vastaavaa ole jo olemassa) ja lisätään siitä suunnattu kaari siihen solmuun, joka LR-jäsentäjässä syötemerkin kanssa määrää uuden tilan. Jos kaksi eri pinoa siirtävät saman tilan pinoon tai redusoituvat samaan tilaan, pinojen pinnat yhdistetään samaksi solmuksi.

5.1. Graafirakenteinen pino (GSS)

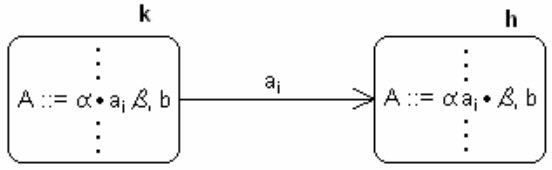
GSS koostuu tilasolmuista, jotka vastaavat tiloja DFA:ssa sekä suunnatuista kaarista. GSS:ään voidaan sisällyttää myös kieliopin symboleja, jos jäsennykset halutaan eksplisiittisesti tallentaa. Tässä esityksessä GSS sisältää myös kieliopin symbolit.

GSS on suunnattu graafi, joka sisältää tilasolmuja ja kieliopin symboleja niin, että kieliopin symbolin sisältävät solmut ja tilan sisältävät solmut vuorottelevat. Tomitan algoritmi saa syötteenään syötemerkkijonon $a_1 a_2 ... a_n$ ja tämän avulla se käy läpi LR(1)-algoritmin muodostamaa determinististä äärellistä automaattia (DFA) rakentaen GSS:ää edetessään. Tällöin GSS sisältää kuvassa 5.1 esitettyä muotoa olevan aligraafin,



Kuva 5.1. Aligraafi GSS:ssä.

jos DFA:ssa on kuvassa 5.2 esitettyä muotoa oleva siirtymä.

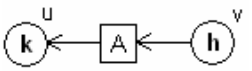


Kuva 5.2. Kuvan 5.1 graafia vastaava DFA.

Rakentamista tehdään askelmissa niin, että alkuaskeleen jälkeen jokaista syötemerkkiä kohti suoritetaan yksi askel. GSS:n tilat kuuluvat joukkoon U. Tämä jaetaan askelien mukaan joukkoihin U_i, $0 \le i \le n$. Kuvassa 5.1 tila k kuuluu siten joukkoon U_{i-1} ja tila h kuuluu joukkoon U_i. Tämä vastaa siis syötemerkin siirtämistä pinoon tavallisessa LR-jäsentäjässä.

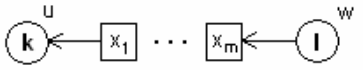
Joukon U_i sanotaan olevan *suljettu redusointien suhteen*, jos joukon kaikista tiloista on suoritettu kaikki mahdolliset redusoinnit. U_0 on alkujoukko, joka sisältää DFA:n alkutilan ja on suljettu redusointien suhteen. Samoin jokainen joukko U_i, $1 \le i \le n$, on muodostettu joukosta U_{i-1} lisäämällä joukkoon U_i tilat, jotka saadaan suorittamalla sallitut siirrä-toiminnot joukon U_{i-1} tiloista ja sitten sulkemalla U_i redusointien suhteen.

GSS sisältää kuvassa 5.3 esitettyä muotoa olevan aligraafin,



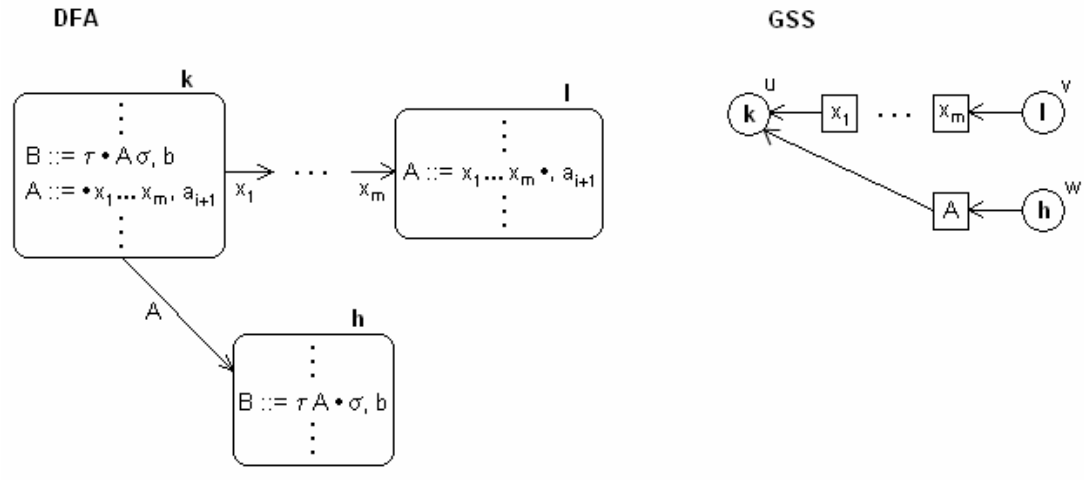
Kuva 5.3. Aligraafi GSS:ssä.

missä $u \in U_j$ ja $v \in U_i$ silloin, kun GSS sisältää myös kuvassa 5.4 esitettyä muotoa olevan aligraafin,



Kuva 5.4. Aligraafi GSS:ssä.

missä $u \in U_j$, $w \in U_i$, $A ::= x_1 ... x_m$ on kieliopin sääntö ja DFA:ssa on siirtymä tilasta k tilaan h symbolilla A. Tämä vastaa siis redusointia tavallisessa LR-jäsentäjässä. Kuvan 5.4 tilanteessa solmun v sanotaan olevan *redusointirelaatiossa* (reduction related) solmuun w (2m+1)-pituisen polun kautta. Kuvassa 5.5 asiaa on vielä havainnollistettu esittämällä vastaava tilanne DFA:ssa.



Kuva 5.5. Redusointirelaatio solmujen v ja w välillä.

Tilasolmun sanotaan olevan tasolla i, jos se kuuluu joukkoon U_i. Tilasolmulla $v \in U_i$ sanotaan olevan *pätevä redusointi*, jos sen sisältämä tila h sisältää DFA:ssa LR-alkion $(A ::= \alpha\bullet, a_{i+1})$. (LR(1)-jäsennyksen mukaisen DFA:n muodostamiseen käytetty algoritmi varmistaa sen, että α on luettu merkkijono.) Vastaavasti, kieliopin säännön $A ::= \alpha$ sanotaan olevan *pätevä tilasolmulle v,* joka on tasolla i ja jonka sisältämä tila on h, jos DFA:ssa tila h sisältää LR-alkion $(A ::= \alpha\bullet, a_{i+1})$. Pätevät redusoinnit suoritetaan siis silloin, kun algoritmi on lukenut merkkijonon $a_1 ... a_i$ ja seuraava syötemerkki on a_{i+1}.

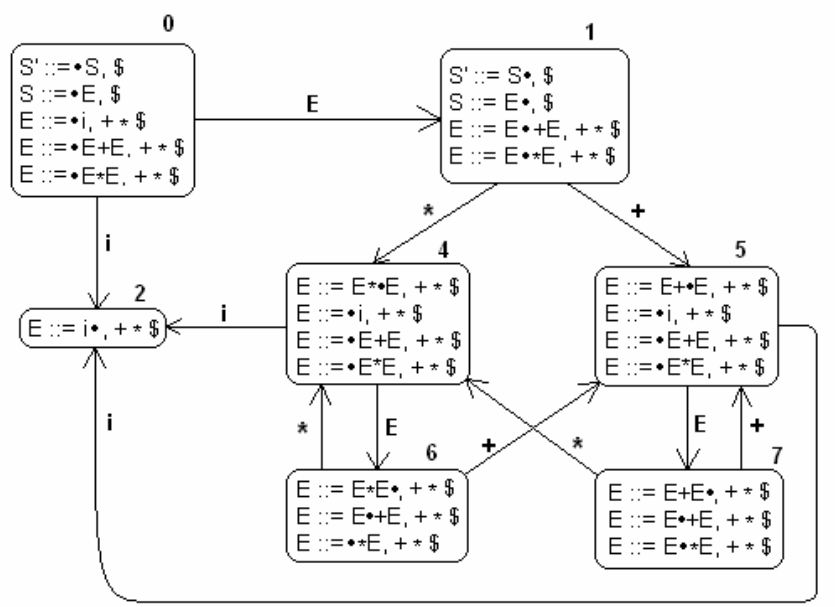
Tilasolmuista koostuva joukko U on suljettu redusointien suhteen, kun sen jokainen taso i on suljettu redusointien suhteen. Oletetaan, että solmu w on joukon U tasolla i ja että se sisältää tilan l. Tarkastellaan nyt tilan l LR(1)-alkiota $(A ::= x_1 ... x_m\bullet, a_{i+1})$. Olkoon k sellaisen solmun u sisältämä tila, joka on saavutettavissa GSS:ssä solmusta w sellaista polkua pitkin, jonka pituus on $2m$. Tällöin DFA:n tilasta k on siirtymä symbolilla A tilaan h. GSS:n joukko U_i sisältää tällöin solmun v tilalla h ja symbolisolmun A, jonka edeltäjä on v ja jälkeläinen on u. Kaikki joukot U_i GSS:ssä ovat suljettuja redusointien suhteen.

5.2. GSS:n rakentaminen syötteestä

Ennen algoritmin formaalia kuvausta esitetään seuraavaksi GSS:n muodostaminen esimerkin avulla. Esimerkkinä käytetään moniselitteistä kielioppia Γ_2, jonka säännöt ovat

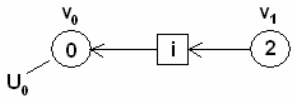
$S ::= E$
$E ::= i$
$E ::= E + E$
$E ::= E * E.$

Ensin muodostetaan LR(1)-alkiojoukkojen mukainen DFA (kuva 5.6).



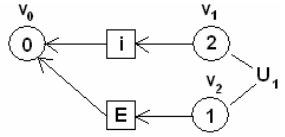
Kuva 5.6. Kieliopista Γ_2 muodostettu LR(1)-jäsennyksen mukainen DFA.

Muodostetaan GSS syötteestä $i + i * i$. Ensimmäisenä joukkoon U_0 lisätään alkutila 0. Tilassa 0 ei ole päteviä redusointeja (kielioppi ei sisällä tyhjiä sääntöjä), joten $U_0 = \{v_0\}$. Tämän jälkeen luetaan merkki i (suoritetaan siirrätoiminto), luodaan symbolisolmu u_1 symbolilla i (merkitään $[u_1, i]$) ja tilasolmu v_1 tilalla 2 (merkitään $[v_1, 2]$). Solmusta v_1 tehdään solmun u_1 edeltäjä ja solmusta u_1 tehdään solmun v_0 edeltäjä. Solmu v_1 lisätään joukkoon U_1 (kuva 5.7).



Kuva 5.7. Syötemerkin *i* lukeminen.

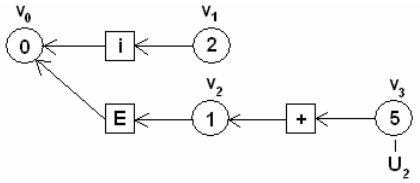
Seuraavaksi tilassa 2 tutkitaan, löytyykö siellä päteviä redusointeja syötemerkin ollessa $+$. Koska tähän mennessä on luettu vain yksi syötemerkki, tälläisen LR(1)-alkion on oltava muotoa $(A ::= a_1 \bullet, a_2)$. Tila 2 sisältää LR(1)-alkion $(E ::= i \bullet, +)$. Tämän mukaan kuljetaan kahden askeleen mittaista polkua solmusta v_1 solmuun v_0. DFA:ssa on siirtymä merkillä E tilasta 0 tilaan 1. Niinpä GSS:ään luodaan tilasolmu $[v_2, 1]$, joka lisätään joukkoon U_1. GSS:ään luodaan myös symbolisolmu symbolilla E, josta tehdään solmun v_2 jälkeläinen ja solmun v_0 edeltäjä (kuva 5.8).



Kuva 5.8. Redusointi sääntöön $E ::= i$ perustuen.

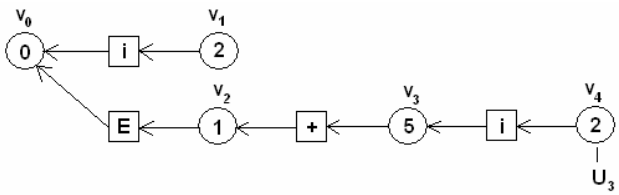
Tämän jälkeen tarkistetaan pätevät redusoinnit juuri lisätystä tilasolmusta ja jatketaan prosessia, kunnes U_1 on suljettu redusointien suhteen. Tässä tapauksessa solmussa v_2 ei ole päteviä redusointeja, joten U_1 on käsitelty.

Yleisesti askeleen *i* alussa GSS:ssä on joukko U_{i-1} tilasolmuja ja syötteenä on $a_i ... a_n \$$. Jokaista joukon U_{i-1} tilasolmua v kohti tarkistetaan, onko DFA:ssa siirtymä merkillä a_i tilasta k tilaan l. Jokaista tälläistä siirtymää kohti tarkistetaan, sisältääkö GSS:n joukko U_i jo tilasolmun tilalla l. Jos ei, lisätään tällainen tilasolmu joukkoon U_i. (Jokainen joukko U_i sisältää siis vain korkeintaan yhden tilan l.) Jos tilalla l on jälkeläisenä symbolisolmu symbolilla a_i, niin tästä symbolisolmusta lisätään kaari solmuun k joukossa U_{i-1}. Muussa tapauksessa luodaan symbolisolmu tilasolmun l jälkeläiseksi symbolilla a_i ja tehdään tästä tilasolmun k edeltäjä. Kun kaikki solmut joukosta U_{i-1} on käsitelty, saadaan $U_i = U_{i-1} a_i$. Joukko U_i saadaan siis joukosta U_{i-1} lukemalla syötemerkki a_i. Esimerkissä askeleella 2 on voimassa $U_1 = \{1,2\}$ ja $U_1 + = \{5\}$ (kuva 5.9).



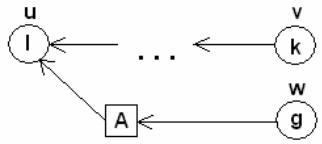
Kuva 5.9. Syötemerkin + lukeminen.

Joukossa $U_2 = \{5\}$ ei ole redusointeja syötemerkillä *i*, joten luetaan seuraava syötemerkki ja lisätään se GSS:ään. Saadaan siis $U_2 i = \{2\}$ (kuva 5.10).



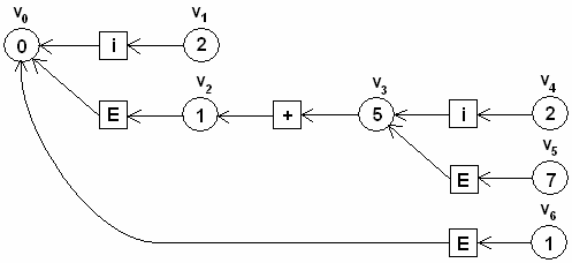
Kuva 5.10. Syötemerkin *i* lukeminen.

Seuraavaksi muodostetaan redusointisulkeuma joukolle $U_{i-1}a_i$. Jokaiselle tilasolmulle $v \in U_i$, jokaiselle tilalle k ja jokaiselle muotoa $(A ::= x_1 ... x_m \bullet, a_{i+1})$ olevalle LR(1)-alkiolle tilassa *k*, etsitään GSS:stä kaikki tilasolmut [*u*, *l*], jotka ovat 2*m* pituisen polun päässä tilasolmusta v. Olkoon DFA:ssa siirtymä tilasta *l* tilaan *g* merkillä A. Jos joukossa U_i ei vielä ole tilasolmua tilalla g, niin luodaan solmu w ja lisätään se joukkoon U_i. Jos tilasolmusta *w* tilasolmuun *u* on jo olemassa kahden askeleen pituinen polku, ei tehdä mitään. Muuten luodaan symbolisolmu *A*, joka on solmun *w* jälkeläinen ja solmun *u* edeltäjä. Prosessia jatketaan, kunnes U_i on suljettu redusointien suhteen (kuva 5.11).



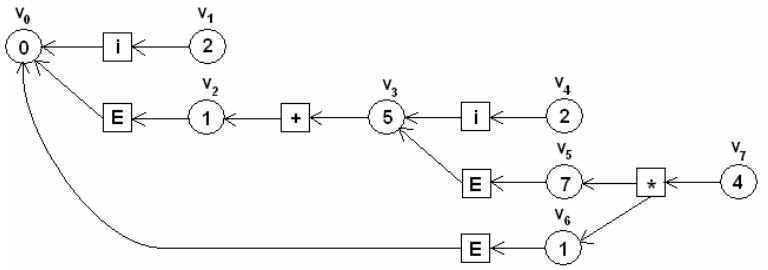
Kuva 5.11. Redusointi GSS:ssä yleisessä tapauksessa.

Esimerkkiin sovellettuna redusointisulkeuman laskeminen tilasta 2 syötemerkillä $*$ lisää joukkoon U_3 tilat 7 ja 1 (kuva 5.12).



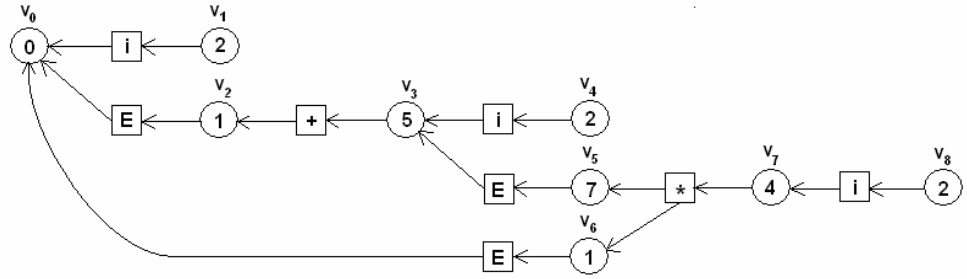
Kuva 5.12. Redusoinnit sääntöihin $E ::= i$ ja $E ::= E + E$ perustuen.

Seuraavaksi luetaan seuraava syötemerkki ∗. DFA:ssa tiloista 1 ja 7 on molemmista siirtymät tilaan 4 syötemerkillä ∗ (kuva 5.13).



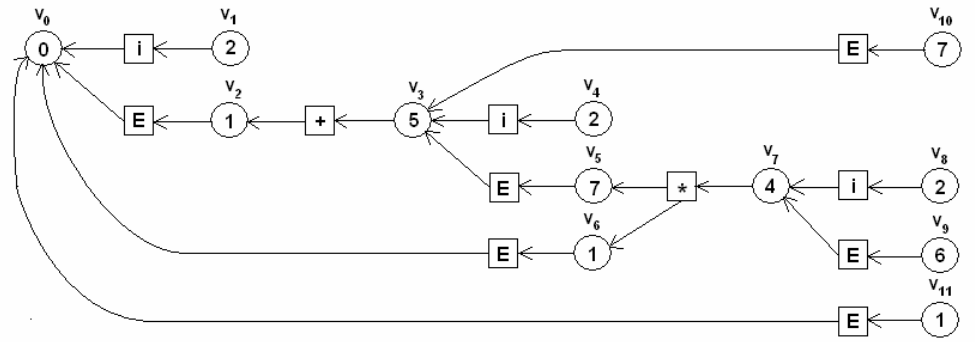
Kuva 5.13. Syötemerkin ∗ lukeminen.

Tilasta 4 ei ole redusointeja syötemerkillä i, joten U_4 = {4}. Luetaan merkki i (kuva 5.14).



Kuva 5.14. Syötemerkin *i* lukeminen.

Tilasta 2 redusoinnin suorittaminen syötemerkillä $ lisää joukkoon U_5 tilan 6. Redusointi tästä lisää joukkoon U_5 tilat 1 ja 7. Redusointi tilasta 7 vie DFA:ssa tilaan 1, mutta koska tämä tila on jo joukossa U_5, ei sitä enää lisätä. Syötemerkin $ lukeminen tilassa 1 johtaa lopulta syötteen hyväksymiseen (kuva 5.15).



Kuva 5.15. Täydellinen GSS syötteelle $i + i * i$.

5.3. Tyhjien sääntöjen käsitteleminen

Kun kieliopissa on tyhjiä sääntöjä, siitä muodostetun LR(1)-jäsennyksen mukaisen DFA:n tiloissa on muotoa $[A ::= \bullet, a]$ olevia LR(1)-alkioita. Jos alkutilassa on LR(1)-alkio $[S ::= \bullet, \$]$, kuuluu tyhjä lause kieliopin määräämään kieleen. Tällaisten redusointien liittäminen yllä kuvattuun GLR-jäsennysmalliin on yksinkertaista.

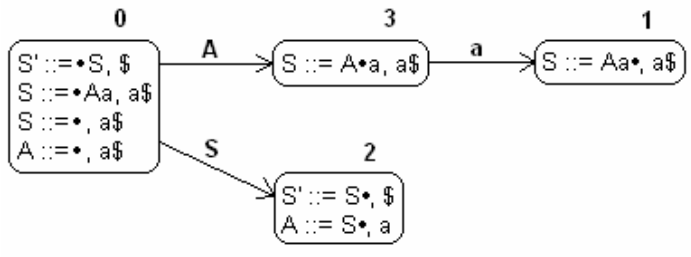
Jos joukossa U on tilasolmu [*v*, *h*] missä DFA:n tila *h* sisältää LR(1)-alkion $[A ::= \bullet, a_{i+1}]$, etsitään LR(1)-jäsennystaulusta riviltä *h*, sarakkeesta *A* kohta *gk*. Tämän jälkeen katsotaan, sisältääkö joukko U_i jo solmun tilalla *k*. Jos ei, luodaan tilasolmu [*w*, *k*] ja symbolisolmu [*u*, *A*]. Solmusta *u* tehdään solmun *w* jälkeläinen ja solmun *v* edeltäjä. Jos joukossa U_i on jo tilasolmu [*w*, *k*], mutta ei kahden askeleen mittaista polkua solmuun *v*, luodaan symbolisolmu [*u*, *A*], ja tämän kautta kahden askeleen mittainen polku solmusta *w* solmuun *v*. Jos joukossa U_i on jo tilasolmu [w, *k*] ja solmusta *w* on kahden askeleen mittainen polku solmuun *v*, ei tehdä mitään.

GSS:ssä on näin polku kahden samaan joukkoon U_i kuuluvan solmun välillä. Tämän seurauksena GSS:ään voi muodostua sykli. Myöhemmin nähdään kuitenkin, että tämä pystytään jäsennyksessä käsittelemään.

Havainnollistetaan seuraavaksi tyhjien sääntöjen käsittelyä GSS:n rakentamisessa. Tarkastellaan kielioppia Γ_3, jonka säännöt ovat

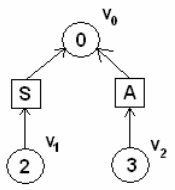
$$S ::= Aa \mid \varepsilon$$
$$A ::= S \mid \varepsilon.$$

Tästä muodostettu LR(1)-jäsennyksen mukainen DFA on esitetty kuvassa 5.16. Syötteenä on *a*.



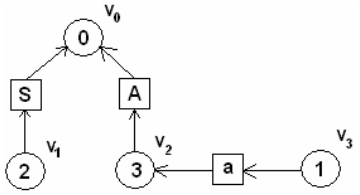
Kuva 5.16. Kieliopista Γ_3 muodostettu LR(1)-jäsennyksen mukainen DFA.

Aluksi GSS:ään laitetaan tilasolmu $[v_0, 0]$. Solmu v_0 asetetaan joukkoon U_0. DFA:n tilassa 0 on kaksi redusointia syötemerkin ollessa *a*. LR(1)-alkio $[S ::= \bullet, a]$ luo tilasolmun $[v_1, 2]$, sekä symbolisolmun symbolilla *S*, josta tehdään solmun v_1 jälkeläinen ja solmun v_0 edeltäjä. LR(1)-alkio $[A ::= \bullet, a]$ luo tilasolmun $[v_3, 3]$, sekä symbolisolmun symbolilla *A*, josta tehdään solmun v_2 jälkeläinen ja solmun v_0 edeltäjä (kuva 5.17).



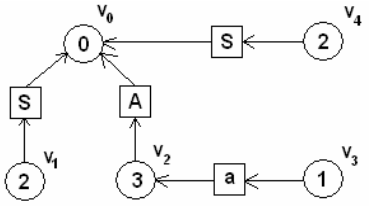
Kuva 5.17. Redusoinnit tyhjiin sääntöihin perustuen.

Solmua v_1 käsiteltäessä huomataan, että DFA:n tilassa 2 on redusointi $[A ::= S\bullet, a]$. Tämä vie takaisin solmuun v_0. Nyt algoritmin mukaan on tarkoitus lisätä GSS:ään ja joukkoon U_0 tilasolmu tilalla 3, sekä GSS:ään tilasolmun jälkeläiseksi symbolisolmu symbolilla *A*. Mutta koska joukossa U_0 on jo tilasolmu tilalla 3 ja GSS:ssä tämän jälkeläisenä symbolisolmu symbolilla *A*, ei tehdä mitään. Solmussa v_2 suoritetaan siirto ja luodaan tilasolmu $[v_3, 1]$, sekä symbolisolmu symbolilla *a*, josta tehdään solmun v_2 jälkeläinen ja solmun v_1 edeltäjä. Solmu v_3 lisätään joukkoon U_1 (kuva 5.18).



Kuva 5.18. Syötemerkin *a* lukeminen.

Syötemerkkinä on nyt *$*. Solmua v_3 käsiteltäessä huomataan, että DFA:n tilassa 1 on redusointi [$S ::= Aa\bullet,\$$]. Tämä vie takaisin solmuun v_0. GSS:ään ja joukkoon U_1 lisätään tilasolmu [v_4, 2], sekä GSS:ään tilasolmun jälkeläiseksi symbolisolmu symbolilla *S*. Mitään muuta ei enää tehdä. Tila 2 on lopputila, joten syöte hyväksytään (kuva 5.19).



Kuva 5.19. Täydellinen GSS syötteelle *a*.

6. Tomitan algoritmi 1e

Tomita [Tom86] kuvasi viisi algoritmia, algoritmit 0, 1, 2, 3 ja 4, yleistettyyn LR-jäsentämiseen. Algoritmi 1 sisältää suurimman osan tarvittavasta välineistöstä, mutta se soveltuu kuitenkin vain kieliopeille, jotka eivät sisällä tyhjiä sääntöjä.

Esitetään seuraavaksi formaalisti menetelmä, jolla tähän mennessä on muodostettu GSS. Kyseessä on Scottin ja muiden [SJH00] tekemä pieni muunnelma Tomitan algoritmista 1. Se suorittaa tyhjiin sääntöihin liittyvät redusoinnit edellä kuvatulla tavalla, kun tilasolmua käsitellään ensimmäisen kerran. Tämä algoritmi on pohjana Scottin ja muiden RNGLR-algoritmille [SJH00], joka esitellään myöhemmin. Lisäksi sen tarkastelu auttaa ymmärtämään, miksi alkuperäiset Tomitan algoritmit 2, 3 ja 4 eivät toimi tietyissä tapauksissa.

Algoritmi ylläpitää seuraavia joukkoja:

A : joukko tilasolmuja, jotka on käsiteltävä GSS:ssä

U_i : joukko askeleen i tilasolmuja

R : joukko suorittamattomia redusointeja

Q : joukko suorittamattomia siirtoja.

Algoritmissa on kolme funktiota, jotka rakentavat GSS:ää:

TOIMIJA: Käsittelee joukkoon A tallennetuttuja tilasolmuja. Lisää joukkoon R redusointeja ja joukkoon Q siirtoja.

REDUSOIJA: Käsittelee redusointeja joukosta R. Lisää tarvittavat tila- ja symbolisolmut GSS:ään.

SIIRTÄJÄ: Käsittelee siirrot joukosta Q. Lisää tarvittavat tila- ja symbolisolmut GSS:ään.

Algoritmin syöte on kontekstiton kielioppi , jonka säännöt on numeroitu, tästä kieliopista muodostettu LR(1)-jäsennyksen mukainen DFA jäsennystaulun muodossa sekä syötemerkkijono $a_i \ldots a_n \$$.

Algoritmi 3. Algoritmi 1e yleistettyyn LR-jäsentämiseen [SJH00].

luo tilasolmu [v_0 , 0], missä 0 on DFA:n alkutila
aseta $U_0 = \{ v_0 \}, A = \theta, R = \theta, Q = \theta$

jokaiselle $i,\ 0 \le i \le n$, suorita JÄSENNÄ_SYMBOLI(i)
olkoon q DFA:n lopputila
jos U_n sisältää tilasolmun, jonka tila on q on DFA:n lopputila, ilmoita jäsennyksen onnistuneen
muuten ilmoita jäsennyksen epäonnistuneen

JÄSENNÄ_SYMBOLI(i) {
  $A = U_i$
  $U_{i+1} = \theta$
  niin kauan kuin $A \neq \theta$ tai $R \neq \theta$ **tee**
    jos $A \neq \theta$ suorita TOIMIJA(i) **muuten** suorita REDUSOIJA(i)
  suorita SIIRTÄJÄ(i)
}

TOIMIJA(i) {
  poista v joukosta A ja olkoon h solmun v tila
  jos 'siirrä k' on toiminta LR(1)-jäsennystaulun kohdassa $(h,\ a_{i+1})$
    lisää (v, k) joukkoon Q
  jokaiselle 'redusoi p' –toiminnalle DFA:n toimintataulun kohdassa $(h,\ a_{i+1})$
    jos säännön p oikean puolen pituus on 0, lisää (v, p) joukkoon R
    muuten jokaiselle solmun v jälkeläissolmulle u, lisää (u, p) joukkoon R
}

REDUSOI(i) {
  poista (u, j) joukosta R
  olkoon m säännön p oikean puolen pituus ja olkoon X säännön p vasemman puolen apumerkki
  jos $m = 0$
    olkoon k solmun u tila ja olkoon gl toiminta LR(1)-jäsennystaulun kohdassa (k, X)
    jos joukossa U_i ei ole tilaa l
      luo GSS:ään uusi tilasolmu [v, l] ja lisää v joukkoihin U_i ja A
    olkoon [v, l] joukon U_i solmu
    jos GSS:ssä on kahden askeleen mittainen polku solmusta v solmuun u, älä tee mitään
    muuten {
      luo GSS:ään symbolisolmu [u', X] ja tee tästä solmun v jälkeläinen ja solmun u edeltäjä
      jos v ei ole joukossa A {
        jokaiselle sellaiselle redusoinnille rk LR(1)-jäsennystaulun kohdassa (l, a_{i+1}), joka ei perustu tyhjään sääntöön
          lisää (u', k) joukkoon R
      }
    }
  muuten
    jokaiselle tilasolmulle w joka voidaan saavuttaa solmusta u $2m-1$ mittaista polkua pitkin **tee** {
      olkoon k solmun w tila ja olkoon gl toiminta LR(1)-jäsennystaulun kohdassa

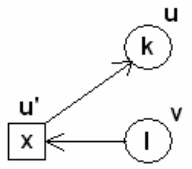
```
            (k, X)
        jos joukossa U_i ei ole tilaa l, luo GSS:ään uusi tilasolmu [v, l] ja lisää v
            joukkoihin U_i ja A
        olkoon [v, l] joukon U_i solmu
        jos GSS:ssä on kahden askeleen mittainen polku solmusta v solmuun u, älä tee
            mitään
        muuten {
          luo GSS:ään symbolisolmu [u', X] ja tee tästä tästä solmun v jälkeläinen ja
            solmun w edeltäjä
          jos v ei ole joukossa A {
            jokaiselle redusoinnille rk DFA:n toimintataulun kohdassa (l, a_{i+1})
              lisää (u', k) joukko on R
          }
        }
      }
}

SIIRTÄJÄ(i} {
  niin kauan kuin Q ≠ θ suorita {
    poista (v, k) joukosta Q
    jos joukossa U_{i+1} ei ole tilasolmua [w, k], luo tälläinen solmu
    jos solmulla w ei ole jälkeläissolmua [u, a_{i+1}], luo tälläinen solmu
    jos solmu u ei ole solmun v edeltäjä, tee solmusta u solmun v edeltäjä
  }
}
```

Funktio TOIMIJA poistaa joukkoon A tallennetun tilasolmun v ja tutkii LR(1)-jäsennystaulusta mahdolliset toiminnot tilasolmun v tilan ja seuraavan syötemerkin a_{i+1} perusteella. Siirrä-toiminnon tapauksessa joukkoon Q lisätään tila v sekä se tila k, johon DFA:ssa siirrytään tilasta v lukemalla a_{i+1}. Redusointien osalta joukkoon R tallennetaan symbolisolmu ja reduoinnissa käytettävän säännön numero. Jos sääntö on tyhjä sääntö, symbolisolmu on v. Muussa tapauksessa joukkoon R tallennetaan jokainen pari (u, p), missä u on solmun v jälkeläinen (siis symbolisolmu) GSS:ssä.

REDUSOI-funktio käsittelee siis paria (u, p), missä u on solmu GSS:ssä ja p on redusoinnissa käytettävän säännön numero. Funktiossa REDUSOIJA redusointien käsittely poikkeaa sen mukaan, onko redusoinninssa käytetty sääntö tyhjä sääntö vai ei (eli onko säännön oikean puolen pituus 0).

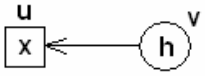
Jos säännön p oikean puolen pituus $m = 0$, niin joukon U_i tilasolmu $[u, k]$, sisältää muotoa $(X ::= \bullet, a_{i+1})$ olevan LR(1)-alkion. Tällöin haetaan LR(1)-taulusta riviltä k, sarakkeesta X kohta gl. Tämän jälkeen tutkitaan, sisältääkö U_i tilasolmun $[v, l]$. Jos ei, tälläinen tilasolmu luodaan GSS:ään ja lisätään

joukkoon U_i. Tämän jälkeen tutkitaan, onko GSS:ssä kahden askeleen pituista polkua tilasolmusta v tilasolmuun u. Jos ei, luodaan symbolisolmu [u', X], josta tehdään solmun v seuraaja ja solmun u edeltäjä (kuva 6.1).



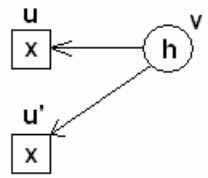
Kuva 6.1. Solmun v lisääminen.

Jos joukon U_i tilasolmu [v, h] sisältää muotoa ($X ::= x_1 ... x_m \bullet, a_{i+1}$) olevan LR-alkion, missä $m \geq 1$, on etsittävä kaikki tilasolmut z, jotka ovat 2m-pituisen polun päässä tilasolmusta v. Ongelmana on kuitenkin se, että myöhemmässä vaiheessa saatetaan solmulle v lisätä uusi jälkeläinen ja luoda näin uusi samanpituinen polku solmusta v sekä uusi mahdollinen redusointi, jota ei olla vielä käsitelty. Jos tilasolmussa joudutaan käymään uudestaan, koska siitä on lisätty uusi polku, niin kaikkien polkujen läpikäyminen tekisi jäsentäjästä hyvin tehottoman. (Tämä ei aiheuttaisi virhettä, koska algoritmissa tarkistetaan kahden askeleen pituisten polkujen olemassaolo. Näin jo tehtyjä redusointeja ei toisteta.) Tämän takia funktiossa TOIMIJA redusoinnit tallennetaan ensimmäisen solmun mukaan, joka on tilasolmusta lähtevän polun varrella. Kun tilasolmu v lisätään GSS:ään, sillä on tila h ja jälkeläisenä symbolisolmu u (kuva 6.2).



Kuva 6.2. Tilasolmu v ja symbolisolmu u.

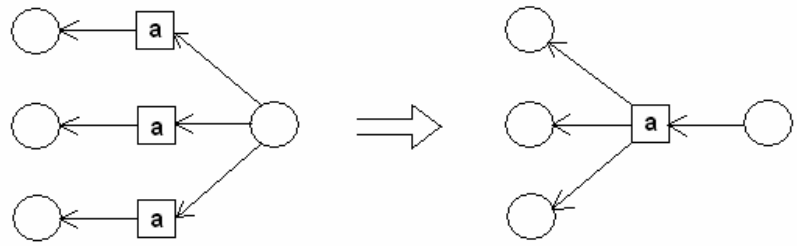
Jokaista tilan h muotoa ($X ::= x_1 ... x_m \bullet, a_{i+1}$) olevaa LR(1)-alkiota kohti REDUSOI-funktion käsiteltäväksi tulee redusointi (u, p), missä p on säännön $X ::= x_1 ... x_m$ numero. Kun redusointia (u, p) käsitellään, etsitään kaikki solmut, jotka ovat (2m-1)-pituisen polun päässä solmusta u. Jos myöhemmässä vaiheessa tilasolmusta v luodaan uusi polku (kuva 6.3),



Kuva 6.3. Tilasolmu *v* ja symbolisolmut *u* ja *u* ‘.

niin kaikille tilan h redusoinneille suoritettaviksi redusoinneiksi määritellään (u', p). Näin jo suoritettuja redusointeja ei enää käydä läpi.

SIIRTÄJÄ-funktiossa yhdistetään perusmerkit niin, että kaikki polut joukon U_i tietystä tilasta joukkoon U_{i-1} kulkevat saman symbolisolmun kautta (kuva 6.4).



Kuva 6.4. Symbolisolmujen yhdistäminen.

6.1. Esimerkki epäsuorasta vasemmasta rekursiosta

Seuraavaksi edellä annettua Tomitan algoritmia havainnollistetaan esimerkin avulla. Esimerkissä syötteenä on merkkijono *xb* ja kielioppina Γ_4 jonka säännöt ovat

$$S' ::= \mathrm{S}$$
$$S ::= ASb \mid x$$
$$A ::= \varepsilon .$$

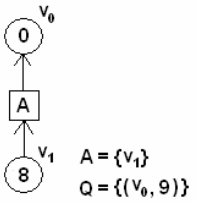
Kielioppi sisältää *epäsuoran vasemman rekursion* (hidden left recursion), koska se sisältää johdon $S \Rightarrow ASb \Rightarrow Sb$. Määritelmän mukaan kielioppi sisältää epäsuoran vasemman rekursion, jos se sisältää sellaisen apumerkin *A* ja sellaiset merkkijonot τ ja σ siten, että $\sigma \overset{+}{\Rightarrow} \varepsilon$ ja $A \overset{*}{\Rightarrow} \sigma A \tau$.

LR(1)-jäsennystaulu kieliopille Γ_4 on esitetty kuvassa 6.5.

	$	a	x	A	S
0			r3/s9	g8	g7
1		r1			
2		s1			
3	r1				
4		s3			
5			r3/s6	g5	g2
6		g2			
7	hyv.				
8			r3/s6	g5	g4
9	r2				

Kuva 6.5. LR(1)-jäsennystaulu kieliopille Γ_4.

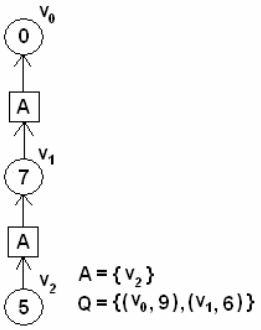
GSS:n muodostaminen aloitetaan lisäämällä tyhjään graafiin tilasolmu [v_0, 0]. Siis A = U_0 = {v_0}, R = Q = θ ja syötemerkkinä on $a_1 = x$. Solmu v_0 poistetaan joukosta A ja tutkitaan LR(1)-taulusta toiminta tilan ollessa 0 ja syötemerkin ollessa x. Toimintoja on kaksi: redusointi säännön 3 ($A ::= \varepsilon$) mukaan ja syötemerkin siirto. Siirto (v_0, 9) asetetaan joukkoon Q. Säännön $A ::= \varepsilon$ oikean puolen pituus on 0, joten suoritettavaksi redusoinniksi joukkoon R tulee (v_0, *8*). Suoritetaan seuraavaksi tämä redusointi. LR(1)-taulussa kohdassa (*0, A*) on toiminto *g8*, joten GSS:n lisätään tilasolmu [v_1, *8*] sekä symbolisolmu symbolilla *A*, josta tehdään solmun v_1 jälkeläinen ja solmun v_0 edeltäjä. Asetetaan solmu v_1 joukkoihin *A* ja U_0 (kuva 6.6).



Kuva 6.6. Redusointi sääntöön $A ::= \varepsilon$ perustuen.

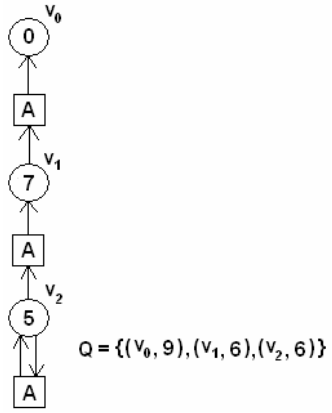
Solmu v_1 poistetaan joukosta *A*. LR(1)-jäsennystaulussa kohdassa (*8, x*) on kaksi toimintoa: redusointi säännön 3 mukaan ja syötemerkin siirto. Siirto (v_1, 6) asetetaan joukkoon *Q* ja redusointi säännön 3 mukaan asetetaan joukkoon *R*. Suoritetaan redusointi, jolloin GSS:ään lisätään tilasolmu [v_2, 5] sekä

symbolisolmu symbolilla A, josta tehdään solmun v_2 jälkeläinen ja solmun v_1 edeltäjä. Solmu v_2 asetetaan joukkoihin A ja U_0 (kuva 6.7).



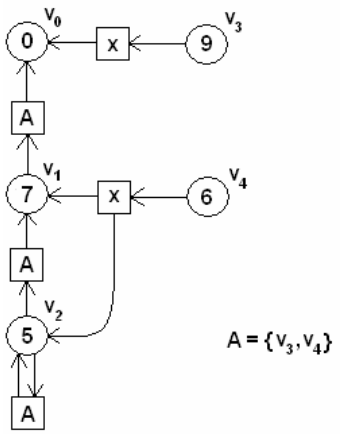
Kuva 6.7. Redusointi sääntöön $A ::= \varepsilon$ perustuen.

Käsiteltäessä joukon A solmua v_2 huomataan, että U_0 sisältää jo tilan 5, joten solmusta v_2 luodaan polku itseensä. Koska tilassa 5 ei ole pituudeltaan nollaa suurempia redusointeja, uusia redusointeja ei enää lisätä. Joukkoon Q lisätään siirto (v_2, 6) (kuva 6.8).



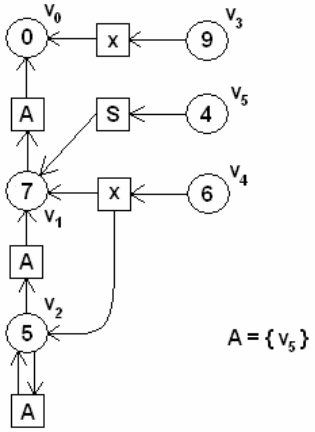
Kuva 6.8. Redusointi sääntöön $A ::= \varepsilon$ perustuen.

Suoritetaan seuraavaksi siirrot joukosta R. GSS:ään ja joukkoon U_1 lisätään tilasolmut [v_3, 9] ja [v_4, 6]. Solmut v_3 ja v_4 lisätään joukkoon A (kuva 6.9).



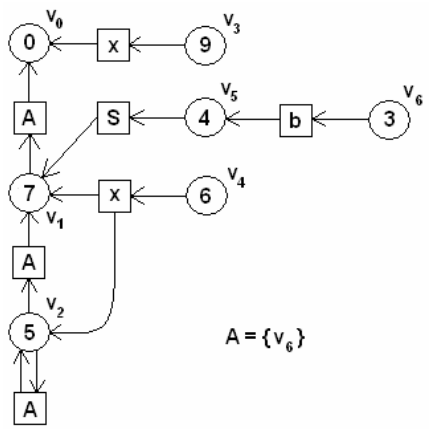
Kuva 6.9. Syötemerkin *x* lukeminen.

Poistetaan joukosta *A* solmu v_3. LR(1)-jäsennystaulussa ei ole toimintoja kohdassa (9, *b*), joten ei tehdä mitään. Seuraavaksi poistetaan solmu v_4 joukosta *A*. Toiminto LR(1)-jäsennystaulun kohdassa (6, *b*) on redusoida säännön 2 mukaan. Solmusta v_4 kuljetaan takaisin solmuun v_1. LR(1)-jäsennystaulun merkintä kohdassa (7, *S*) on *g4*, joten lisätään joukkoon U_1 tilasolmu [v_5, 4]. GSS:ään lisätään myös symbolisolmu symbolilla *S*, josta tehdään solmun v_5 jälkeläinen ja solmun v_1 edeltäjä. Solmu v_5 asetetaan joukkoon *A* (kuva 6.10).



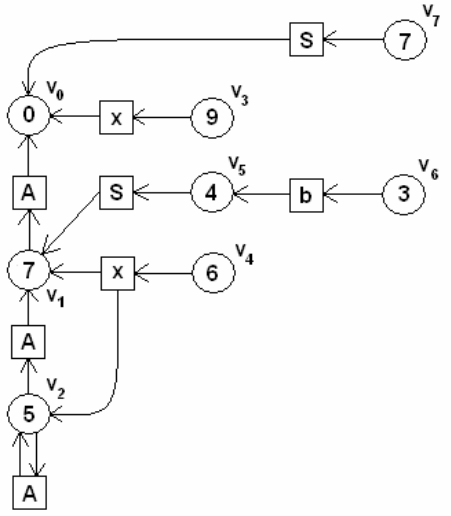
Kuva 6.10. Redusointi sääntöön $S ::= x$ perustuen.

Käsitellään joukon *A* solmu v_5. Ainoa toiminto LR(1)-jäsennystaulun kohdassa (4, *b*) on syötemerkin siirto, joten joukkoon *Q* lisätään alkio (v_5, 3). Joukot *A* ja *R* ovat tyhjiä, joten suoritetaan joukosta *Q* sinne juuri lisätty siirto. Lisätään siis GSS:ään tilasolmu [v_6, 3], joka lisätään joukkoon U_2 ja *A* (kuva 6.11).



Kuva 6.11. Syötemerkin *b* lukeminen.

Käsitellään joukon *A* solmu v_6. LR(1)-jäsennystaulun kohdassa (3, $) on redusointi sääntöön 1 perustuen. Suoritetaan tämä redusointi, jolloin kuljetaan takaisin solmuun v_0 ja katsotaan toiminto LR(1)-jäsennystaulun kohdasta (0, *S*). Toiminnon *g7* mukaan lisätään joukkoon U_2 tilasolmu [v_7, 7]. Muita toimintoja ei tämän jälkeen tehdä. Joukko U_2 sisältää nyt tilasolmun [v_7, 7], joka on lopputila. Näin syöte *xb* hyväksytään (kuva 6.12).



Kuva 6.12. Täydellinen GSS syötteelle *xb*.

6.2. Esimerkki epäsuorasta oikeasta rekursiosta

Edellä annettu algoritmi pysähtyy kaikilla syötteillä, koska joukko U_i on äärellinen. Se nimittäin sisältää korkeintaan niin monta tilaa kuin LR(1)-jäsennystaulu sisältää ja mistä tahansa solmusta joukossa U_i on korkeintaan yksi kahden askeleen mittainen polku mihin tahansa solmuun joukossa U_j, $i \le j$. Mutta kuten seuraavaksi huomataan, algoritmi voi hylätä syötteen, joka kuuluu kieliopin määrittelemään kieleen.

Tarkastellaan kielioppia Γ_5, jonka säännöt ovat

$S' ::= S$
$S ::= \text{a S A} \mid \varepsilon$
$A ::= \varepsilon$.

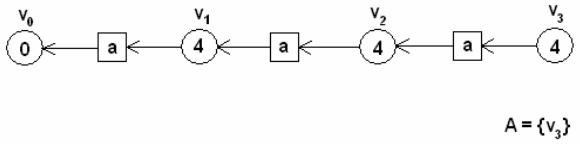
Kielioppi sisältää *epäsuoran oikean rekursion* (hidden right recursion), koska se sisältää säännön $S \Rightarrow aSA \Rightarrow aS$. Määritelmän mukaan kielioppi sisältää epäsuoran oikean rekursion, jos se sisältää sellaisen apumerkin A ja merkkijonot τ ja σ, että $\tau \overset{+}{\Rightarrow} \varepsilon$ ja $A \overset{*}{\Rightarrow} \sigma A \tau$. LR(1)-jäsenystaulu kieliopille Γ_5 on esitetty kuvassa 6.13.

	$	a	A	S
0	r2	s4		g3
1	r1			
2	r3		g1	
3	hyv.			
4	r2	s4		g2

Kuva 6.13. LR(1)-jäsennystaulu kieliopille Γ_5.

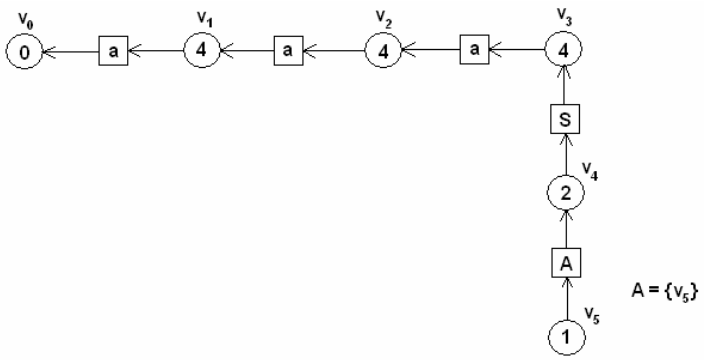
LR(1)-jäsennystaulusta huomataan, että Γ_5 on LR(1)-kielioppi, koska LR(1)-jäsennystaulussa ei ole konflikteja. Syötemerkkijonona on *aaa*.

Syötemerkillä *a* ei ole redusointeja, joten kaikki merkit *a* lukemalla päästään kuvan 6.14 GSS:ään.



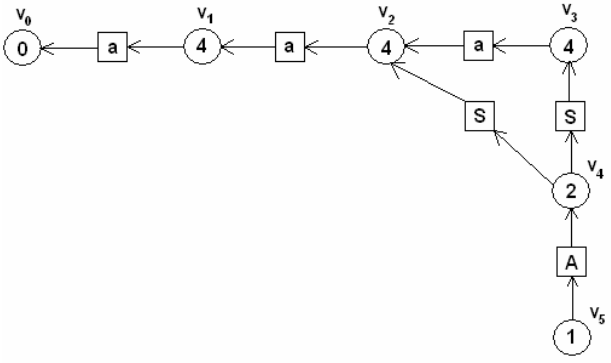
Kuva 6.14. Merkkijonon *aaa* lukeminen.

Kuvan 6.14 tilanteessa on voimassa $U_0 = \{v_0\}$, $U_1 = \{v_1\}$, $U_2 = \{v_2\}$ ja $v_3 \in U_3$. Seuraavaksi muodostetaan redusointisulkeuma joukolle U_3 syötemerkillä \$ eli poistetaan v_3 joukosta *A* ja käsitellään se. Toiminta *r2* luo uuden tilasolmun [v_4, 2], sekä kahden askeleen mittaisen polun solmusta v_4 solmuun v_3. Solmu v_4 lisätään joukkoihin U_3 ja *A*. Solmu v_4 poistetaan joukosta *A* ja käsitellään, jolloin toiminta *r3* luo uuden tilasolmun [v_5, 1]. Solmu v_5 lisätään joukkoihin U_3 ja *A* (kuva 6.15).



Kuva 6.15. Redusoinnit sääntöihin $S ::= \varepsilon$ ja $A ::= \varepsilon$ perustuen.

Poistetaan solmu v_5 joukosta *A* ja käsitellään se. Redusointi säännön 1 mukaan vie tilasolmuun [v_2, 4]. Kohta (*4*, *S*) LR(1)-jäsennystaulussa on *g2*. Joukko U_3 sisältää jo tilasolmun [v_4, 2], joten solmusta v_4 luodaan kahden askeleen mittainen polku solmuun v_2 (kuva 6.16).



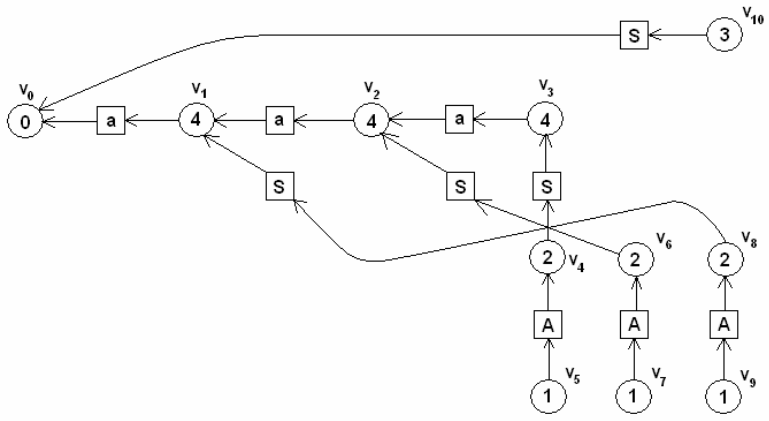
Kuva 6.16. Redusointi sääntöön $S ::= aSA$ perustuen.

Algoritmin mukaan seuraavaksi solmusta v_2 kaikki pituudeltaan nollaa suuremmat redusoinnit lisätään joukkoon *R*. Koska LR(1)-taulun kohdassa (*2*,

$) ei tällaisiä redusointeja ole, joukkoon *R* ei lisätä mitään. Joukot *A* ja *R* ovat tyhjiä, joten algoritmi pysähtyy. Koska U_3 ei sisällä lopputilaa, syöte *aaa* hylätään, vaikka se kuuluu kieliopin Γ_5 määrittelemään kieleen.

Ongelmana on nyt se, että uusi polku luotiin keskelle olemassaolevaa polkua, eli uusi kaari lisättiin lähteväksi solmusta, jolla oli jo edeltäjä. Näin tilasolmusta v_5 lähtevää uutta polkua ei käsitelty (, mikä lopulta olisi johtanut merkkijonon *aaa* hyväksymiseen).

Tomitan ratkaisu tähän ongelmaan oli jakaa joukko U_i alijoukkoihin silloin, kun suoritetaan redusointi tyhjään sääntöön perustuen. Tällöin esimerkissä, jossa solmut v_4 ja v_5 ovat luotu tyhjään sääntöön perustuvan redusoinnin kautta, solmut v_3, v_4 ja v_5 kuuluvat eri joukkoihin. Niinpä suoritettaessa redusointia solmusta v_5, GSS:ään lisätään solmu v_6, koska siinä joukossa, jossa solmu v_5 on, ei ole tilasolmua tilalla 2. Solmussa v_6 suoritetaan taas redusointi tyhjään sääntöön perustuen, joten luotava solmu v_7 kuuluu eri joukkoon kuin v_6. Suorittamalla redusointi solmusta v_7, kuljetaan solmuun v_1, luodaan solmu v_8 ja kahden askeleen mittainen polku solmusta v_8 solmuun v_1. Solmu v_8 luo solmun v_9, josta suoritettava redusointi lopulta luo tilasolmun [v_{10}, 3], joka on lopputila (kuva 6.17).



Kuva 6.17. Tomitan algoritmin 2 syötteestä *aaa* muodostama GSS.

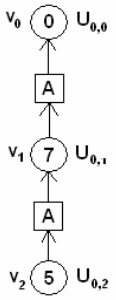
Alijoukkojen luominen joukolle U_i johtaa kuitenkin siihen, että näin toimiva algoritmi ei pysähdy käsitellessään kielioppia, joka sisältää epäsuoran vasemman rekursion.

6.3. Tomitan algoritmi 2

Esitetään seuraavaksi esimerkki siitä, kuinka Tomitan algoritmi käyttäytyy epäsuoran vasemman rekursion sisältävällä kieliopilla. Aiheeseen palataan tarkemmin luvussa 9.

Perusideana on siis toimia samoin kuin yllä 1ε-algoritmissa, mutta kaksi tilasolmua *a* ja *b* samalla algoritmin askeleella *i* kuuluvat eri alijoukkoihin $U_{i,j}$ ja $U_{i,j+1}$, jos solmu *b* on luotu tekemällä tyhjään sääntöön perustuva redusointi solmussa *a*. Tarkastellaan jälleen kielioppia Γ_4 syötteellä *aa*.

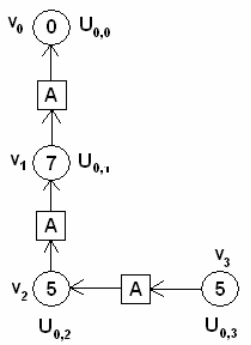
Suoritettaessa pelkkiä redusointeja alkutilan 0 sisältävästä tilasolmusta v_0 päädytään kuvan 6.18 tilanteeseen.



Kuva 6.18. Redusoinnit sääntöön $A ::= \varepsilon$ perustuen.

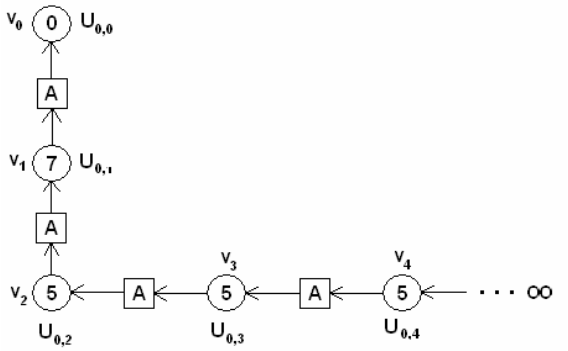
Solmu v_0 kuuluu joukkoon $U_{0,0}$. Tästä suoritettu redusointi tyhjän säännön $A ::= \varepsilon$ perusteella luo uuden tilasolmun [v_1, 7], joka kuuluu joukkoon $U_{0,1}$. Suorittamalla redusointi solmusta v_1 jälleen säännön $A ::= \varepsilon$ mukaan, luodaan tilasolmu [v_2, 5]. Solmu v_2 kuuluu joukkoon $U_{0,2}$, koska jälleen on suoritettu redusointi tyhjään sääntöön perustuen.

Toiminta LR(1)-jäsennystaulun kohdassa (5, *a*) on redusoida säännön $A ::= \varepsilon$ mukaan. Toiminta LR(1)-jäsennystaulun kohdassa (5, *A*) on *g5*, joten luodaan uusi tilasolmu [v_3, 5], joka lisätään joukkoon $U_{0,3}$, koska on suoritettu redusointi tyhjään sääntöön perustuen (kuva 6.19).



Kuva 6.19. Redusointi sääntöön $A ::= \varepsilon$ perustuen.

Suorittamalla redusointi solmusta v_3 joudutaan jälleen luomaan uusi alijoukko $U_{0,4}$, johon laitetaan tilasolmu [v_4, 5]. Koska alijoukot sisältävät tilan 5, josta redusoimalla joudutaan luomaan uusi alijoukko tilalla 5, ei tämä prosessi pääty koskaan (kuva 6.20).



Kuva 6.18. Päättymätön silmukka Tomitan algoritmissa 2.

7. Farshin algoritmi

Naivi ratkaisu tilanteeseen, jossa uusi polku luodaan keskelle olemassaolevaa polkua, on tutkia uudestaan kaikki solmut joukosta U_i, jotka eivät ole enää joukossa A. Näistä etsitään redusoinnit, jotka voivat käyttää alipolkunaan uutta polkua. Farshin [NF91] muunnos Tomitan algoritmista käyttää tätä menetelmää.

Algoritmi 4. Farshin yleistetty LR-jäsennysalgoritmi kontekstittomille kieliopeille [NF91].

```
JÄSENNÄ(G, a_1...a_n ) {
  Γ := θ
  a_{n+1:} = $
  r := EPÄTOSI
  luo solmu v_0 merkillä s_0
  U_0 := {v_0}
  jokaiselle i, 0 ≤ i ≤ n, suorita JÄSENNÄ_SANA
  palauta r
}

JÄSENNÄ_SANA {
  A := U_i
  R := θ, Q := θ
  toista
    jos A ≠ θ suorita TOIMIJA muuten jos R ≠ θ suorita VIIMEISTELIJÄ
  kunnes R = θ ja A = θ
  suorita SIIRTÄJÄ
}

TOIMIJA {
  poista joukosta A elementti v
  jokaiselle α LR(1)-jäsennystaulun kohdassa (TILA(v), a_{i+1}) {
    jos α = 'hyväksy' r := TOSI
    jos α = 'siirrä s' lisää (v, s) joukkoon Q
    jos α = 'redusoi p' {
      olkoon m säännön p oikean puolen pituus
      jokaiselle sellaiselle solmulle w, joka 2m pituisen polun päässä solmusta v
        /* tyhjille säännöille w = v */
        lisää (w, p) joukkoon R
    }
  }
}
```

```
VIIMEISTELIJÄ {
  poista pari (w, p) joukosta R
  olkoon N säännön p oikean puolen pituus
  olkoon s merkintä LR(1)-jäsennystaulun kohdassa (TILA(w), N)
  jos on olemassa sellainen u ∈ U_i, että TILA(u) = s {
    jos solmusta u ei ole kahden askeleen mittaista polkua solmuun w
      luo GSS:ään solmu z merkillä N
      tee solmusta z solmun u jälkeläinen ja solmun w edeltäjä
      jokaiselle v ∈ (U_i \ A)
        jokaiselle sellaiselle säännölle q, että 'redusoi q' on LR(1)-jäsennystaulun
                kohdassa (TILA(v), a_{i+1})
          olkoon m säännön q oikean puolen pituus
          jokaiselle sellaiselle solmulle t niin, että solmusta v on 2m askeleen
                pituinen polku solmun z kautta solmuun t
            lisää (t, q) joukkoon R
  }
  muuten {
    luo GSS:ään tilasolmu [ u, s] ja symbolisolmu [z, N]
    tee solmusta z solmun u jälkeläinen ja solmun w edeltäjä
    lisää u joukkoihin A ja U_i
  }
}

SIIRTÄJÄ {
  U_{i+1} := θ
  niin kauan kuin Q ≠ θ {
    poista pari (v, s) joukosta Q
    luo GSS:ään symbolisolmu x symbolilla a_{i+1}
    tee solmusta x solmun v edeltäjä
    jos on olemassa sellainen u ∈ U_{i+1}, että TILA(u) = s
      tee solmusta x solmun u jälkeläinen
    muuten {
      luo tilasolmu u tilalla s ja tee solmusta x solmun u jälkeläinen
      lisää u joukkoon U_{i+1}
    }
  }
}
```

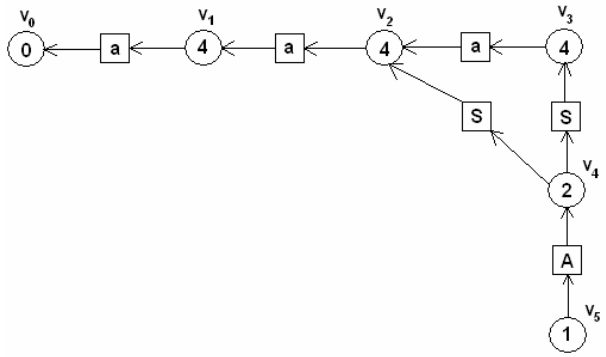
Farshin algoritmi perustuu Tomitan algoritmiin 1, mutta tyhjiä sääntöjä käsiteltäessä algoritmi luo syklejä GSS:ään, aivan kuten algoritmissa 1e tehdään. Mutta toisin kuin Tomitan algoritmeissa ja algoritmissa 1e joukkoon *R* tallennetaan pareja eri periaatteella. Farshin algoritmissa joukkoon *R* lisättävä pari sisältää suoraan sen tilasolmun, josta redusointi suoritetaan. Muistetaan, että Algoritmissa 1e tämä tilasolmu etsittiin vasta funktiossa REDUSOIJA. Joukkoa *R* käsittelevän funktion Farshi nimesi VIIMEISTELIJÄ:ksi (completer).

Tarkastellaan funktion VIIMEISTELIJÄ kohtaa, jossa luodaan kahden askeleen mittainen polku olemassaolevasta tilasolmusta.

jos on olemassa sellainen $u \in U_i$, että TILA(u) = s {
  jos solmusta u ei ole kahden askeleen mittaista polkua solmuun w
    luo GSS:ään solmu z merkillä N
    tee solmusta z solmun u jälkeläinen ja solmun w edeltäjä
    jokaiselle $v \in (U_i \setminus \mathrm{A})$
      jokaiselle sellaiselle säännölle q, että 'redusoi q' on LR(1)-
          jäsennystaulun kohdassa (TILA(v), a_{i+1})
        olkoon m säännön q oikean puolen pituus
        jokaiselle sellaiselle solmulle t niin, että solmusta v on $2m$ askeleen
            pituinen polku solmun z kautta solmuun t
          lisää (t, q) joukkoon R
}

Tässä tutkitaan kaikki nykyisen joukon U_i solmut, jotka on jo käsitelty (ne eivät ole enää joukossa A). Näistä solmuista tarkastetaan, voidaanko suorittaa sellaisia redusointeja, joissa kuljetaan uuden symbolisolmun z kautta. Kaikki ne tilasolmut, joihin päästään redusoinnissa kulkemalla tämän symbolisolmun kautta tallennetaan joukkoon R redusoinnissa käytettävän säännön q kanssa.

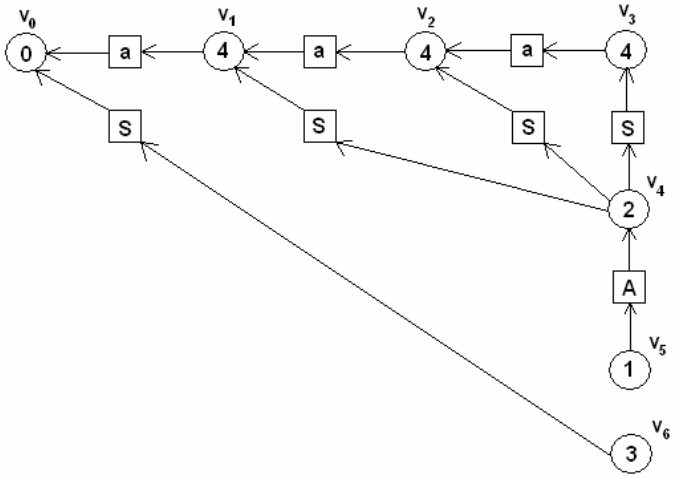
Tarkastellaan kieliopia Γ_3 syötteellä *aaa*. Kuvan 7.1 tilanteessa,



Kuva 7.1. Solmusta v_5 on suoritettu redusointi sääntöön $S ::= aSA$ perustuen.

jossa tilasolmusta v_4 on luotu kahden askeleen mittainen polku tilasolmuun v_2, tutkitaan uudestaan tilasolmut v_3, v_4 ja v_5. Näistä solmuista solmussa v_5 on redusointi, jossa kuljetaan solmusta v_4 solmun v_2 kautta solmuun v_1 ja luodaan kahden askeleen mittainen polku solmusta v_4 solmuun v_1. Jälleen on luotu uusi polku keskelle olemassaolevaa polkua, joten tutkitaan uudestaan solmut v_3, v_4 ja v_5. Solmussa v_5 on redusointi, jossa kuljetaan alipolun

solmusta v_4 solmun v_1 kautta solmuun v_0. Nyt luodaan joukkoon U_3 tilasolmu [v_6, 3]. Tila 3 on lopputila, joten syöte *aaa* hyväksytään (kuva 7.2).



Kuva 7.1. Farshin algoritmin muodostama täydellinen GSS syötteelle *aaa*.

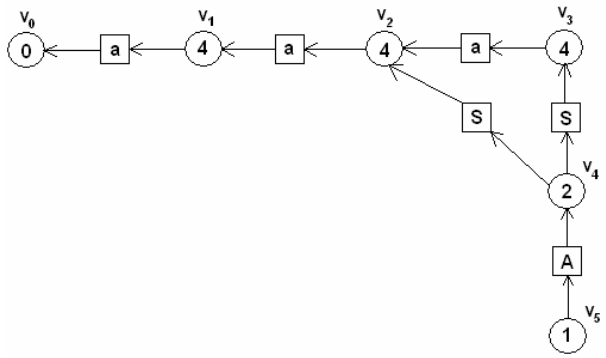
Farshi ei esitä formaalia todistusta algoritminsa toimivuudesta. Farshin algoritmi muodostaa kuitenkin perustan teolliselle sovellukselle [vdBHdJ+ar] joten se on luotettavasti testattu käytännössä [JS02].

8. RNGLR-algoritmi

Scott ja muut [SJH00] huomauttavat, että Farshin raa'an voiman lähestymistapa, jossa kaikki solmut joukossa $U_i \setminus A$ on tutkittava uudelleen, on tehoton. Kaikista solmuista on etsittävä redusoinnit ja näistä redusoinneista polut, jotka kulkevat uuden polun kautta. Scott ja muut esittävät sen sijaan algoritmiin 1e perustuvan RNGLR-algoritmin, jossa LR(1)-jäsennystaulua on muunneltu ja se käsittelee valmiina muotoa $(A ::= x_1 ... x_j \bullet \beta, a)$ olevia LR(1)-alkioita, missä $\beta \overset{*}{\Rightarrow} \varepsilon$.

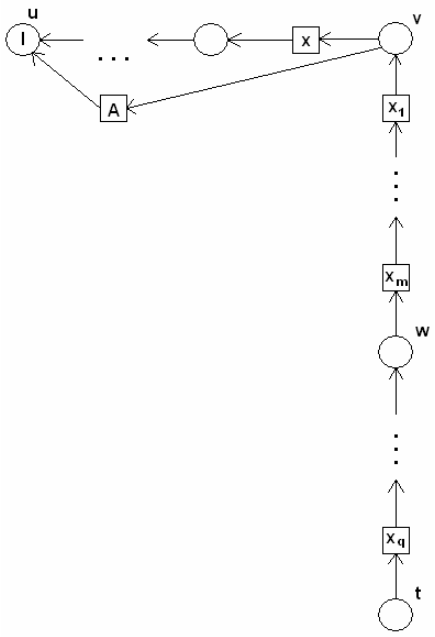
8.1. Epäsuora oikea rekursio ja oikeanpuoleisesti tyhjentyvät säännöt

Analysoidaan seuraavaksi syitä siihen, miksi 1e-algoritmi hylkäsi edellä kieleen kuuluvan lauseen. Uusi polku lisättiin keskelle olemassaolevaa polkua, eli luotiin polku tilasolmusta v_4, jolla oli edeltäjä tilasolmu v_5. Polun luonnin aiheutti tässä tapauksessa solmussa v_5 ollut redusointi. Koska redusoinnin jälkeen v_5 oli käsitelty, ei voitu enää suorittaa solmusta v_5 redusointia, joka olisi kulkenut uuden polun kautta (kuva 8.1).



Kuva 8.1. Solmusta v_5 on suoritettu redusointi sääntöön $S ::= aSA$ perustuen.

Yleisessä tapauksessa tilanne on kuvan 8.2 kaltainen.



Kuva 8.2. Uuden polun lisääminen keskelle olemassaolevaa polkua.

Tilasolmusta t on suoritettu redusointi ja tilasolmusta v luodaan kahden askeleen mittainen polku tilasolmuun u, jossa keskellä olevan symbolisolmun symboli on A. Niinpä tilasolmun t tilassa oleva redusointi perustuu muotoa $A ::= ...xx_1...x_m...x_q$ olevaan sääntöön.

Algoritmin 1e ominaisuuksien mukaan tilasolmusta voidaan luoda lähtevä kaari vain, jos tilasolmu kuuluu nykyiseen joukoon U_i. Niinpä tilasolmun v on kuuluttava nykyiseen joukkoon U_i. Kaikki solmut, joista on polku tilasolmuun v, on täytynyt lisätä tämän jälkeen, joten nämäkin kuuluvat joukkoon U_i. Scott ja muut [SJH00] ovat osoittaneet, että jos GSS:ssä on kahden askeleen pituinen polku kahden samaan joukkoon U_i kuuluvan tilasolmun välillä, tämä polku on tehty sääntöön $C ::= \gamma$ perustuvan redusoinnin kautta, missä $\gamma \overset{*}{\Rightarrow} \varepsilon$. Niinpä jokaisella tilasymbolilla x_i, $1 \le i \le q$, jotka ovat polulla tilasolmusta t tilasolmuun v, on johto $x_i \overset{*}{\Rightarrow} \varepsilon$.

Niin ikään algoritmin 1e ominaisuuksien mukaan kaikilla tietyn tilasolmun jälkeläisinä olevilla symbolisolmulla on oltava sama symboli. Niinpä kuvassa 8.2 x = A. Näin ollen solmun t tilassa oleva redusointi perustuu muotoon $A ::= ...Ax_1...x_m...x_q$ olevaan sääntöön, missä siis $x_i \overset{*}{\Rightarrow} \varepsilon$, $1 \le i \le q$. Tämä sääntö sisältää siis epäsuoran oikean rekursion ja juuri tämä saa aikaan sen, että 1e-algoritmi hylkää kieleen kuuluvan lauseen.

8.2. GLR-jäsentäminen muunnellulla DFA:lla

Esitetään seuraavaksi Scottin ja muiden tekemä muutos 1e-algoritmiin. Sääntöjä, jotka ovat muotoa $A ::= \alpha\beta$ ja $\beta \overset{+}{\Rightarrow} \varepsilon$, sanotaan *oikealta tyhjentyviksi* (right nullable). Kielioppi on oikealta tyhjentyvä, jos se sisältää oikealta tyhjentyviä sääntöjä. Scottin ja muiden *RNGLR*-algoritmissa (right nullable GLR) algoritmissa redusointeja suoritetaan myös silloin, kun viimeinen ei-tyhjentyvä symboli on luettu. Jos siis DFA:n tilassa h on LR(1)-alkio $(A ::= x_1 ... x_m B_1 ... B_n, b)$, missä $n = 0$ tai $B_i \overset{*}{\Rightarrow} \varepsilon$, $1 \le i \le n$, ja säännön $A ::= x_1 ... x_m B_1 ... B_n$ numero on p, LR(1)-toimintataulun kohdassa (h, a) on toiminta (*rp*, *m*)

Syötteenä algoritmi saa kontekstittoman kieliopin, jonka säännöt on numeroitu, tästä kieliopista muodostettu redusointi-muunneltu LR(1)-jäsennyksen mukainen DFA LR(1)-jäsennystaulun muodossa, sekä syötemerkkijonon $a_i ... a_n \$$.

Algoritmi 5. Scottin ja muiden yleistetty LR-jäsennysalgoritmi kontekstittomille kieliopeille [SJH00].

```
JÄSENTÄJÄ {
  luo tilasolmu v_0 DFA:n alkutilalla 0
  aseta U_0 = {v_0}, R = θ, Q = θ, a_{n+1} = $, U_1 = θ, ..., U_n = θ
  jos LR(1)-jäsennystaulun kohdassa (0, a_1) on toiminta sk
    lisää (v_0, k) joukkoon Q
  jokaiselle toiminnalle (rp, 0) LR(1)-jäsennystaulun kohdassa (0, a_1)
    lisää (v_0, X, 0) joukkoon R, missä X on säännön p vasen puoli
  jokaiselle i, 0 ≤ i ≤ n, niin kauan kuin U_i ≠ θ suorita {
     niin kauan kuin R ≠ θ suorita REDUSOIJA(i)
     suorita SIIRTÄJÄ(i)
  }
  jos U_n sisältää DFA:n lopputilan, hyväksy syöte, muuten hylkää syöte
}

REDUSOI(i) {
  poista (v, X, m) joukosta R
  etsi tilasolmuista koostuva joukko χ, joka voidaan saavuttaa solmusta v 2(m-1)-
pituista polkua pitkin, tai nollan askeleen pituista polkua pitkin, jos m = 0
  jokaiselle tilasolmulle u ∈ χ {
    olkoon k solmun u tila ja olkoon gl merkintä LR(1)-jäsennystaulun kohdassa (k, X)
    jos joukossa U_i on tilasolmu [w, l] {
      jos solmusta w solmuun u ei ole kahden askeleen mittaista polkua {
```

```
        jos solmusta w on kahden askeleen mittainen polku solmuun joukossa U_i
          tee tämän polun keskimmäisestä solmusta solmun u edeltäjä
        muuten luo symbolisolmu symbolilla X ja tee tästä solmun w jälkeläinen ja
                solmun u edeltäjä
        jos m ≠ 0 {
          jokaiselle toiminnalle (rp, t) LR(1)-jäsennystaulun kohdassa (l, a_{i+1}), missä
                t ≠ 0 ja B on säännön p vasen puoli
            lisää (u, B, t) joukkoon R
        }
      }
      muuten {
        luo uusi tilasolmu w GSS:ään tilalla l ja uusi symbolisolmu [y, X]
        tee solmusta y solmun w jälkeläinen ja solmun u edeltäjä
        lisää w joukkoon U_i
        jos LR(1)-jäsennystaulun kohdassa (l, a_{i+1}) on toiminta sh
          lisää (w, h) joukkoon Q
        jokaiselle toiminnalle (rj, 0) LR(1)-jäsennystaulun kohdassa (l, a_{i+1})
          lisää (w, B, 0) joukkoon R, missä B on säännön p vasen puoli
        jos m ≠ 0 {
          jokaiselle toiminnalle (rp, t) LR(1)-jäsennystaulun kohdassa (l, a_{i+1}), missä
                t ≠ 0 ja B on säännön p vasen puoli
          lisää (u,B, t) joukkoon R
      }
    }
  }

SIIRTÄJÄ(i) {
  jos i ≠ n {
    Q'= θ /*väliaikainen joukko uusien siirtojen säilyttämiseen*/
    niin kauan kuin Q = θ suorita {
      poista elementti (v, k) joukosta Q
      jos joukossa U_{i+1} on tilasolmu [w, k] {
        olkoon u symbolisolmu, joka on solmun w jälkeläinen
        tee solmusta u solmun v edeltäjä
        jokaiselle toiminnalle (rp, t) muunnellun LR(1)-jäsennystaulun kohdassa
                (k, a_{i+2}), missä t ≠ 0
          lisää (v, B, t) joukkoon R, missä B on säännön p vasen puoli
      }
      muuten {
        luo GSS:ään uusi tilasolmu [w, k] ja uusi symbolisolmu [u, a_{i+1}]
        tee solmusta u solmun w jälkeläinen ja solmun v edeltäjä
        lisää solmu w joukkoon U_{i+1}
        jos muunnellun LR(1)-jäsennystaulun kohdassa (k, a_{i+2}) on toiminto sh
          lisää (w, h) joukkoon Q'
        jokaiselle toiminnolle (rp, t) muunnellun LR(1)-jäsennystaulun kohdassa
                (k, a_{i+2}), missä t ≠ 0
```

```
            lisää (v, B, t) joukkoon R, missä B on säännön p vasen puoli
          jokaiselle toiminnolle (rp, 0) muunnellun LR(1)-jäsennystaulun kohdassa
                    (k, a_{i+2})
            lisää (v, B, 0) joukkoon R, missä B on säännön p vasen puoli
      }
      kopioi Q' joukkoon Q
  }
}
```

RNGLR-algoritmissa ei ole erikseen funktiota TOIMIJA, joka käsittelee joukon A tilasolmuja, vaan kaikki tarvittavat siirrot ja redusoinnit määritellään tilasolmujen luontien yhteydessä.

Tilasolmuista suoritettavat redusoinnit on suoritettava kaikkia $2m$-pituisia polkuja pitkin. Redusoinnit tallennetaan muodossa (v, X, m), missä v on solmu, josta polut etsitään ja X on redusoinnissa käytettävän säännön vasen puoli. Jos $m = 0$, on polkuja on vain yksi, eli tyhjä polku. Jos $m \geq 1$, niin aina lisättäessä solmulle v jälkeläissolmua, luodaan polut pituudeltaan $2m$ solmusta v. Jos uusi solmu w luodaan tai olemassaolevalle solmulle w luodaan uusi jälkeläinen tyhjään sääntöön perustuvan redusoinnin seurauksena, kaikki nollaa askelta pidemmät redusoinnit on jo suoritettu solmun w jälkeläistilasolmussa. Tästä pitää huolen muunnellussa LR(1)-jäsennystaulussa olevat redusoinnit oikealta tyhjentyville säännöille. Siksi nollaa askelta pidempiä redusointeja ei tällaisissa tapauksissa kirjata suoritettavaksi joukkoon R.

Jos uusi solmu w luodaan tai olemassaolevalle solmulle w luodaan uusi jälkeläinen siirron tai sellaisen redusoinnin seurauksena, joka ei perustu tyhjään sääntöön, niin kaikki sellaiset redusoinnit solmusta w, jotka eivät perustu tyhjiin sääntöihin, kirjataan joukkoon R muodossa (v, B, t), missä v on toinen tilasolmu polulla, joka alkaa solmusta w.

8.3. Esimerkki RNGLR-algoritmin toiminnasta

Havainnollistetaan seuraavaksi RNGLR-algoritmin toimintaa kieliopilla, jossa edellä huomattiin 1e-algoritmin hylkäävän lauseen, joka kuului kieliopin määrittelemään kieleen. Käytetään siis kielioppia Γ_5, jonka säännöt ovat

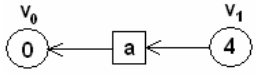
$$S ::= \mathrm{a\ S\ A} \mid \varepsilon$$
$$A ::= \varepsilon.$$

Redusointi-muunneltu LR(1)-jäsennystaulu kieliopille Γ_5 on esitetty kuvassa 8.3. Syötemerkkijonona on jälleen *aaa*.

	$	a	A	S
0	(r2,0)	s4		g3
1	(r1,3)			
2	(r1,2)/(r3,0)		g1	
3	hyv.			
4	(r1,1)/(r2,0)/s4			g2

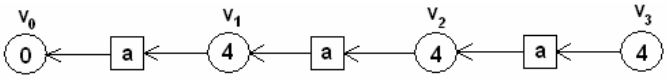
Kuva 8.3. Redusointi-muunneltu LR(1)-jäsennystaulu kieliopille Γ_5.

Aloitetaan luomalla tilasolmu [v_0, 0]. Muunnellun LR(1)-jäsennystaulun kohdassa (*0, a*) on toiminta *s4*, joten luodaan tilasolmu [v_1, 4] sekä symbolisolmu symbolilla *a*, josta tehdään solmun v_1 jälkeläinen ja solmun v_0 edeltäjä (kuva 8.4).



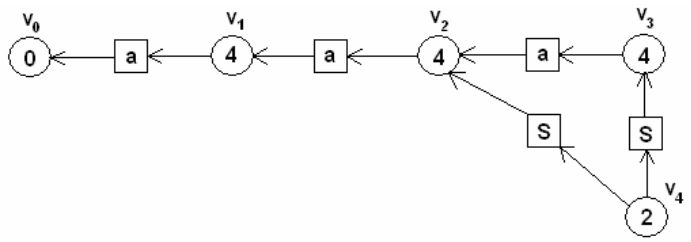
Kuva 8.4. Syötemerkin *a* lukeminen.

Luetaan tämän jälkeen kaksi seuraavaa syötemerkkiä *a* ja luodaan tilasolmut [v_2, 4] ja [v_3, 4]. Tilasolmun v_3 luonnin yhteydessä muunnellun LR(1)-jäsennystaulun kohdan (*4, $*), jossa on toiminnat (*r1, 1*) ja (*r2, 0*), mukaan lisätään joukkoon *R* redusoinnit (v_2, *S, 1*) ja (v_3, *S, 0*) (kuva 8.5).



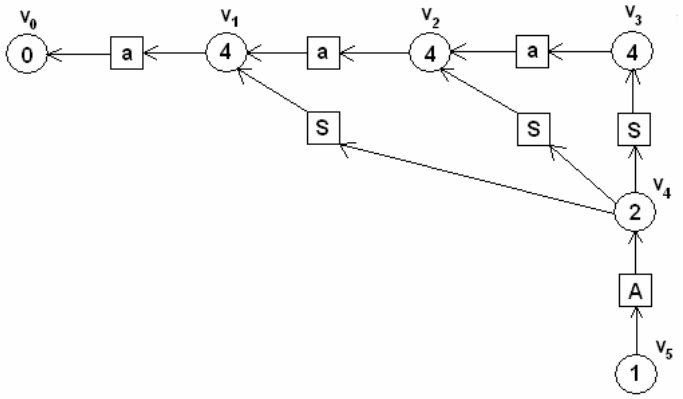
Kuva 8.5. Syötemerkkijonon *aa* lukeminen.

Käsitellään seuraavaksi nämä redusoinnit, joista redusointi (v_2, *S, 1*) luo GSS:ään uuden tilasolmun [v_4, 2] sekä kahden askeleen mittaisen polun solmusta v_4 solmuun v_2. Solmun v_4 luonnissa joukkoon *R* lisätään redusoinnit (v_4, *A, 0*) ja (v_2, *S, 2*). Redusointia (v_3, *S, 0*) suoritettaessa huomataan, että tilasolmu tilalla *2* on jo olemassa, joten luodaan kahden askeleen mittainen polku solmusta v_4 solmuun v_3 (kuva 8.6).



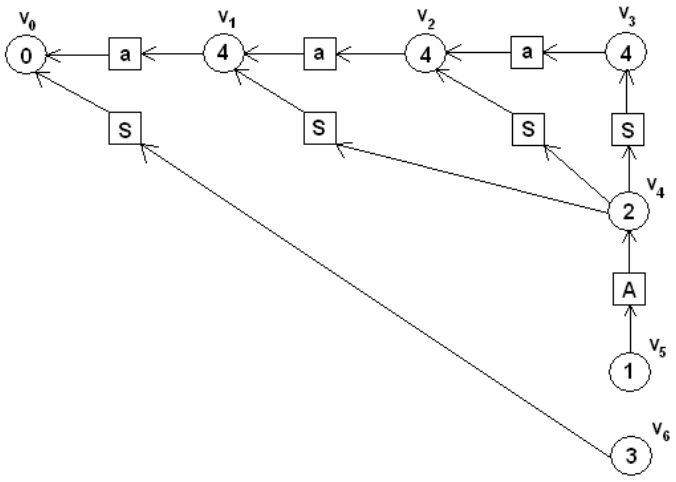
Kuva 8.6. Redusointi oikealta tyhjentyvään sääntöön $S ::= aS$ perustuen.

Redusointi (v_4, A, 0) lisää GSS:ään uuden tilasolmun [v_5, 1] sekä kahden askeleen mittaisen polun solmusta v_5 solmuun v_4. Redusoinnissa (v_2, S, 2) kuljetaan solmusta v_2 solmuun v_1. U_3 sisältää jo tilasolmun tilalla 2, joten solmusta v_4 luodaan kahden askeleen mittainen polku solmuun v_1. Joukkoon R lisätään redusointi (v_1, S, 2) (kuva 8.7).



Kuva 8.7. Redusointi oikealta tyhjentyvään sääntöön $S ::= aS$ perustuen.

Lopuksi redusoinnissa (v_1, S, 2) kuljetaan solmusta v_1 solmuun v_0 ja luodaan joukkoon U_3 uusi tilasolmu [v_6, 3] (kuva 8.8).



Kuva 8.8. RNGLR-algoritmin muodostama täydellinen GSS syötteelle *aaa*.

Mitään toimintoja ei voida enää tehdä. Joukko U_3 sisältää DFA:n lopputilan, joten syöte *aaa* hyväksytään.

9. Tomitan algoritmi 2

Palataan seuraavaksi Tomitan algoritmiin 2. Edellä huomattiin algoritmin joutuvan päättymättömään silmukkaan epäsuoran vasemman rekursion sisältävillä kieliopeilla. Annetaan ensin algoritmin formaali kuvaus.

9.1. Tomitan algoritmi 2 tyhjiä sääntöjä sisältäville kieliopeille

Tomitan algoritmi 2 on sellainen laajennos Tomitan algoritmiin 1, että se käsittelee myös tyhjiin sääntöihin perustuvia redusointeja. Algoritmissa on erillinen funktio E-REDUSOIJA näitä redusointeja varten. Algoritmin aluksi ensimmäinen tilasolmu v_0 asetetaan joukkoon $U_{0,0}$. Algoritmin askeleella i syötesymbolin a_i siirto luo joukkoon $U_{i,0}$ tilasolmun, ellei joukossa $U_{i,0}$ jo ole tilasolmun tilaa. Kaikki askeleella i suoritettavat redusoinnit, jotka eivät perustu tyhjään sääntöön, lisäävät myös tilasolmun joukkoon $U_{i,0}$, mikäli tilasolmun tilaa ei tässä joukossa ole. Suoritettaessa tyhjään sääntöön perustuvaa redusointia funktio E-REDUSOIJA lisää tilasolmun joukkoon $U_{i,j+1}$ (mikäli joukossa $U_{i,j+1}$ ei ole tilasolmun tilaa). Jokainen algoritmin *i*. askeleella suoritetusta tyhjään sääntöön perustuvasta redusoinnissa luotu tilasolmu kuuluu siis eri joukkoon $U_{i,j}$.

Algoritmi ylläpitää seuraavia joukkoja, jotka eivät esiintyneet algoritmissa 1e tai niiden rooli on hiukan muuttunut.

$U_{i,j}$: Joukko tilasolmuja, jotka on luotu jäsennettäessä syötemerkkiä a_i. Jos askeleella *i* ei ole tilasolmua luotaessa suoritettu yhtään tyhjään sääntöön perustuvaa redusointia, j = 0. Jokainen tyhjään sääntöön perustuva redusointi kasvattaa indeksiä j yhdellä. Joukkoa $U_{i,j}$ käsitellään funktioissa TOIMIJA, REDUSOIJA ja E-REDUSOIJA.

R_ε: Joukko tilasolmuja, joista on suoritettava tyhjään sääntöön perustuva redusointi. Jokainen joukon R_ε alkio on pari (v, p), missä $v \in U_{i,j}$ ja p on tyhjä sääntö. Solmulle v suoritetaan toiminto'redusoi p'. Redusoinnin suorittaa funktio E-REDUSOIJA. Joukko R_ε alustetaan funktiossa JÄSENNÄ_SYMBOLI ja sitä käsitellään myös funktiossa TOIMIJA.

Q: Joukko tilasolmuja, joissa on suoritettava siirto. Jokainen joukon Q alkio on pari (v,k), missä $v \in U_{i,0} \cup U_{i,1} \cup ... \cup U_{i,j}$ ja k on tilan numero.

Solmulle v suoritetaan toiminto 'siirrä k'. Siirron suorittaa funktio SIIRTÄJÄ. Joukko Q alustetaan funktiossa JÄSENNÄ_SYMBOLI ja sitä käsitellään myös funktiossa TOIMIJA.

Algoritmin syöte on kontekstiton kielioppi , jonka säännöt on numeroitu, tästä kieliopista muodostettu LR(1)-jäsennyksen mukainen DFA jäsennystaulun muodossa sekä syötemerkkijono $a_i ... a_n \$$.

Algoritmi 6. Tomitan yleistetty LR-jäsennysalgoritmi kontekstittomille kieliopeille [Tom86].

luo tilasolmu [v_0, 0], missä 0 on DFA:n alkutila
aseta $U_{0,0} = \{v_0\}$, $A = \theta$, $R = \theta$, $R_\varepsilon = \theta$, $Q = \theta$
jokaiselle i, $0 \le i \le n$, **suorita** JÄSENNÄ_SYMBOLI(i)
olkoon q DFA:n lopputila
jos $U_{n,i}$, $0 \le i \le j$ sisältää tilasolmun, jonka tila on q, ilmoita jäsennyksen onnistuneen
muuten ilmoita jäsennyksen epäonnistuneen

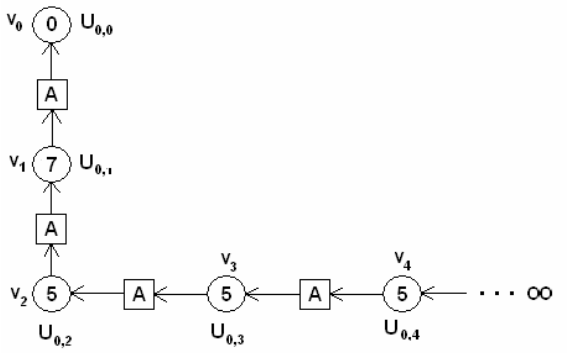
```
JÄSENNÄ_SYMBOLI(i) {
  A = U_{i,0}
  U_{i+1,0} = θ
  niin kauan kuin A ≠ θ tai R ≠ θ tai R_ε ≠ θ
    jos A ≠ θ suorita TOIMIJA(i)
    muuten jos R ≠ θ suorita REDUSOIJA(i)
    muuten jos R_ε ≠ θ suorita E-REDUSOIJA(i)
}

TOIMIJA(i) {
  poista solmu v joukosta A ja olkoon h solmun v tila
  jos 'siirrä k' on toiminta LR(1)-jäsennystaulun kohdassa (h, a_{i+1})
    lisää (v, k) joukkoon Q
  jokaiselle 'redusoi p' toiminnolle LR(1)-jäsennystaulun kohdassa (h, a_{i+1}),
            missä sääntö p on ei-tyhjä
    jokaiselle solmun v jälkeläissolmulle u
      lisää (u, p) joukkoon R
  jokaiselle 'redusoi p' toiminnolle LR(1)-jäsennystaulun kohdassa (h, a_{i+1}),
            missä sääntö p on tyhjä
    lisää (v, p) joukkoon R_ε
}

REDUSOIJA(i) {
  poista (u, p) joukosta R
  olkoon m säännön p oikean puolen pituus ja olkoon X
       säännön p vasemman puolen symboli
```

```
  jokaiselle tilasolmulle w joka voidaan saavuttaa solmusta u
       (2m-1) pituista polkua pitkin suorita {
    olkoon k solmun w tila ja olkoon gl toiminta LR(1)-jäsennystaulun
      kohdassa (k, X)
    jos joukossa U_{i,j} ei ole tilasolmua tilalla l, luo uusi tilasolmu [v, l]
          ja lisää v joukkoihin U_{i,j} ja A
    olkoon [v, l] tilasolmu joukosta U_{i,j}
    jos GSS:ssä on olemassa kahden askeleen mittainen polku solmusta v
          solmuun w, älä tee mitään
    muuten {
      luo GSS:ään uusi symbolisolmu [u', X]
      tee symbolisolmusta u' solmun v seuraaja ja solmun w jälkeläinen
      jos v ei ole joukossa A
        jokaiselle redusoinnille rq LR(1)-jäsennystaulun kohdassa (l, a_{i+1}), missä q ei
              ole tyhjä sääntö
          lisää (u', q) joukkoon R
      }
    }
  }
}

E-REDUSOIJA(i) {
  U_{i,j+1} = θ
  jokaiselle alkiolle (v, p) joukossa R_ε {
    olkoon X säännön p vasemman puolen symboli ja olkoon h solmun v tila. Olkoon
          gl merkintä LR(1)-jäsennystaulun kohdassa (h, X)
    jos joukossa U_{i,j+1} on [w, l]
      luo GSS:ään uusi symbolisolmu [u, X] ja tee solmusta u solmun w
          jälkeläinen ja solmun v edeltäjä
    muuten {
      luo GSS:ään tilasolmu [w, l] ja symbolisolmu [u, X]
      tee symbolisolmusta u solmun w seuraaja ja solmun v jälkeläinen
      lisää w joukkoon U_{i,j+1}
    }
    aseta R_ε = θ, A = U_{i,j+1}, j = j + 1
  }
}

SIIRTÄJÄ(i) {
  niin kauan kuin Q ≠ θ suorita {
    poista elementti (v, k) joukosta Q
    jos joukossa U_{i+1,0} ei ole tilasolmua [w, k], luo tälläinen solmu
    jos solmulla w ei ole jälkeläissolmua [u, a_{i+1}], luo tälläinen solmu
    jos solmu u ei ole solmun v edeltäjä, tee solmusta u solmun v edeltäjä
  }
}
```

9.2. Tomitan algoritmi 2 ja epäsuora vasen rekursio

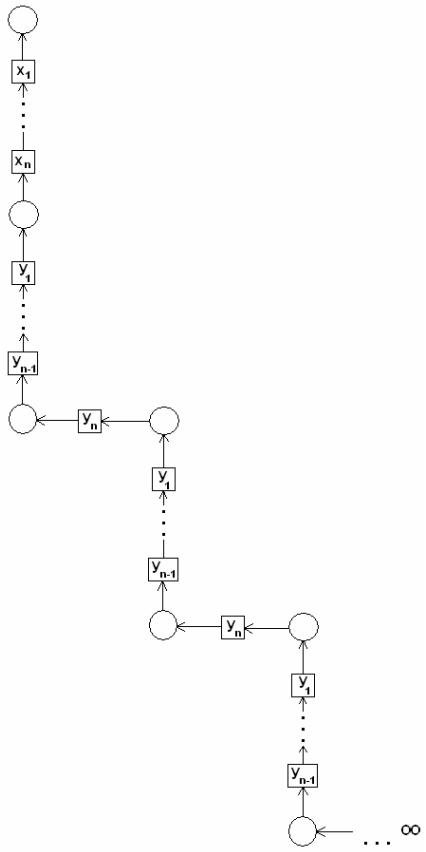
Tomitan algoritmissa 2 askeleella i joukosta $U_{i,j}$ lisätään siis uusi joukko $U_{i,j+1}$ silloin, kun joukossa $U_{i,0}$ suoritetaan redusointi tyhjään sääntöön perustuen. Tällä menettelyllä on tarkoitus estää tilanteet, jossa hylätään lause, joka kuuluu kieliopin määräämään kieleen. Kuten edellä huomattiin, tälläiset tilanteet ovat mahdollisia kieliopeilla, jotka sisältävät epäsuoran oikean rekursion. Mutta, kuten edellä myös huomattiin, jakamalla joukkoa $U_{i,j}$ uusiin joukoihin, algoritmi ei aina pysähdy.

Luvun 6 esimerkissä päädyttiin kuvan 9.1 tilanteeseen.



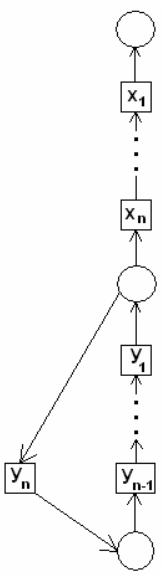
Kuva 9.1. Päättymätön silmukka Tomitan algoritmissa 2.

Yleisesti ottaen tilanne on kuvan 9.2 kaltainen.



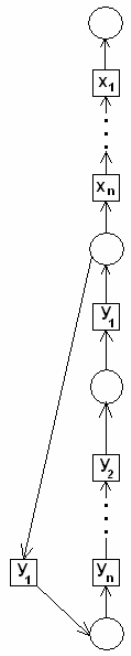
Kuva 9.2. Päättymätön silmukka Tomitan algoritmissa 2 yleisessä tapauksessa.

Saman jäsennyksen omaava GSS voidaan myös kuvassa 9.3 esittetyssä muodossa.



Kuva 9.3. Sykli GSS:ssä.

Koska GSS:n ominaisuuksien perusteella jokainen tilasolmusta lähtevä symbolisolmu on samanmerkkinen, täytyy olla $x_m = y_n$. GSS voidaan siten esittää kuvan 9.4 muodossa.



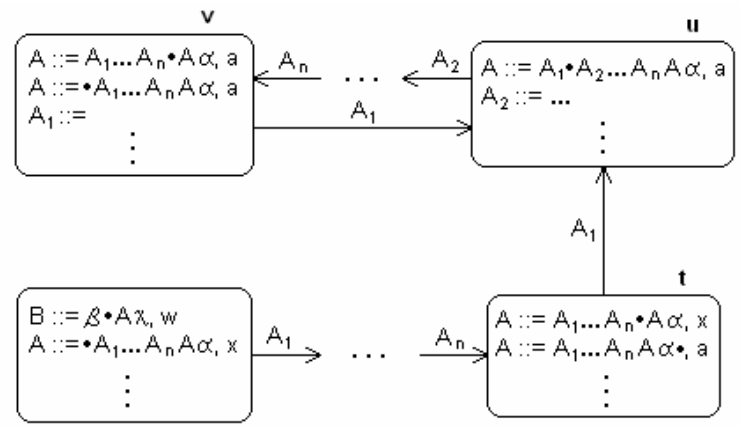
Kuva 9.4. Sykli GSS:ssä.

GSS:stä nähdään, että kieliopissa on rekursiivinen johto ja jonoa $y_1 \ldots y_n$ voidaan siis toistaa loputtomiin. Kaikki kuvan 9.4 tilasolmut on luotu samalla algoritmin askeleella i. Näin ollen $x_i \overset{*}{\Rightarrow} \varepsilon$, $1 \le i \le m$ ja kieliopin täytyy sisältää epäsuora vasen rekursio.

Apumerkki A sisältää epäsuoran vasemman rekursion, jos kieliopissa on oikealta tyhjentyvä sääntö $A ::= \alpha A \beta$. Edellä nähtiin, että Tomitan algoritmissa 2 epäsuoran vasemman rekursion sisältävä kielioppi vei GSS:n rakentamisessa tilanteeseen, jossa prosessia jatkettiin loputtomiin. Tämä johtui syklistä joka GSS:ään muodostuu, kun sitä rakennetaan vasemmanpuoleisen epäsuoran rekursion sisältävän kieliopin LR(1)-jäsennystaulun mukaan niin, että jakoa alijoukkoihin U_i ei tehdä. Tämä taas johtuu syklistä, joka esiintyy LR(1)-jäsennyksen mukaisessa DFA:ssa.

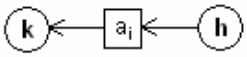
Apulause 1 *Jos kieliopissa* Γ *on muotoa* $A ::= A_1 \ldots A_n A \alpha$ *oleva sääntö ja* A on saavutettavissa kieliopin alkumerkistä, *LR(1)-jäsennyksen mukainen DFA sisältää syklin* $A_1 \ldots A_n$.

Todistus LR(1)-alkiojoukoista muodostetussa LR(1)-jäsennyksen mukaisessa DFA:ssa on tila s, jossa on muotoa $[B ::= \beta \bullet A\chi, w]$ oleva alkio. LR(1)-alkiojoukkoja muodostavan algoritmin sulkeumaoperaatio takaa sen, että tilassa s on myös alkio $[A ::= \bullet A_1 \ldots A_n A\alpha, x]$, missä x = FIRST(χw). DFA:ssa on tällöin siirtymä symboleilla A_1,..., A_n tilaan t, jossa on LR(1)-alkio $[A ::= A_1 \ldots A_n \bullet A\alpha, x]$. Sulkeumaoperaatio takaa, että tilassa t myös on LR(1)-alkio $[A ::= \bullet A_1 \ldots A_n A\alpha, a]$, missä a = FIRST(αx). Tilasta t on DFA:ssa siirtymä tilaan u symbolilla A_1. Tilassa u on LR(1)-alkio $[A ::= A_1 \bullet A_2 \ldots A_n A\alpha, a]$ sekä sulkeumaoperaation muodostamat LR(1)-alkiot. Tilasta u on siirtymät symboleilla $A_2 \ldots A_n$ tilaan v, missä on LR(1)-alkio $[A ::= A_1 \ldots A_n \bullet A\alpha, a]$, sekä sulkeumaoperaation kautta alkio $[A ::= \bullet A_1 \ldots A_n A\alpha, a]$ (koska yllä FIRST(αx) = a, täytyy olla FIRST(αa) = a). Tilasta v on oltava siirtymä symbolilla A_1 sellaiseen tilaan, jossa on LR(1)-alkio $[A ::= A_1 \bullet A_2 \ldots A_n A\alpha, a]$ sekä sulkeumaoperaation muodostamat LR(1)-alkiot. Tila u on tälläinen tila, joten tilasta v on siirtymä symbolilla A_1 tilaan u. Koska tilasta u on siirtymät symboleilla $A_2 \ldots A_n$ tilaan v, sisältää DFA syklin symboleilla $A_1 \ldots A_n$ (kuva 9.5).□



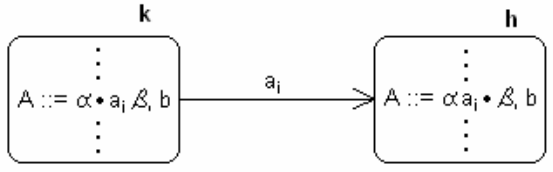
Kuva 9.5. Sykli DFA:ssa merkeillä $A_1...A_n$.

Kuten yllä on todettu, GSS sisältää kuvassa 9.6 esitetyn aligraafin,



Kuva 9.6. Aligraafi GSS:ssä.

jos DFA:ssa on kuvan 9.7 siirtymä.

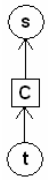


Kuva 9.7. Siirtymä DFA:ssa.

Teoreema 1 *Jos kieliopissa on vasemman rekursion sisältävä apumerkki, joka on saavutettavissa kieliopin alkumerkistä, Tomitan algoritmi 2 joutuu päättymättömään silmukkaan käsitellessään sellaista LR(1)-jäsennyksen mukaista DFA:n tilaa, joka sisältää muotoa* $[A ::= \bullet A_1...A_n A\alpha, x]$, *missä* $A_i \overset{*}{\Rightarrow} \varepsilon$, *olevan LR(1)-alkion.*

Todistus Jos DFA:n tilassa s on muotoa $[A ::= \alpha \bullet B\beta, a]$ oleva LR(1)-alkio, ja $B \overset{*}{\Rightarrow} \varepsilon$, niin LR(1)-alkiojoukkojen muodostamisessa käytettävä sulkeumaoperaatio takaa, että DFA:n tilassa s on myös LR(1)-alkio $[C ::= \bullet\varepsilon, b]$, missä $B \Rightarrow^* C$ ja b = FIRST(βa). Niinpä, jos ollaan tilassa s ja syötemerkki on b,

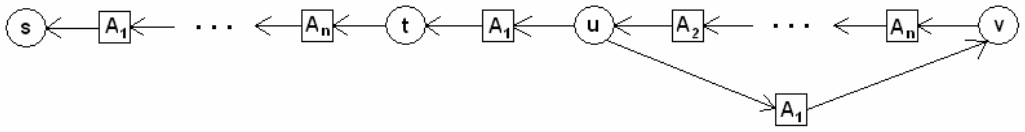
suoritetaan redusointi säännön $C ::= \varepsilon$ perusteella. Rakennettaessa GSS:ää algoritmin 1e mukaan, GSS:ssä on kuvassa 9.8 esitetty aligraafi,



Kuva 9.8. Aligraafi GSS:ssä.

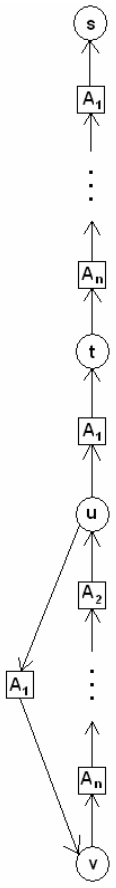
missä LR(1)-jäsennystaulussa on $t = (s, C)$.

Jos DFA:n tilassa s on muotoa $[A ::= \bullet A_1 ... A_n A\alpha, x]$ oleva LR(1)-alkio, missä $A_i \overset{*}{\Rightarrow} \varepsilon$, $1 \le i \le n$, niin apulauseen 1 perusteella GSS:ään tulee kuvassa 9.9 esitetty aligraafi, kun GSS:ää rakennetaan algoritmin 1e mukaan.



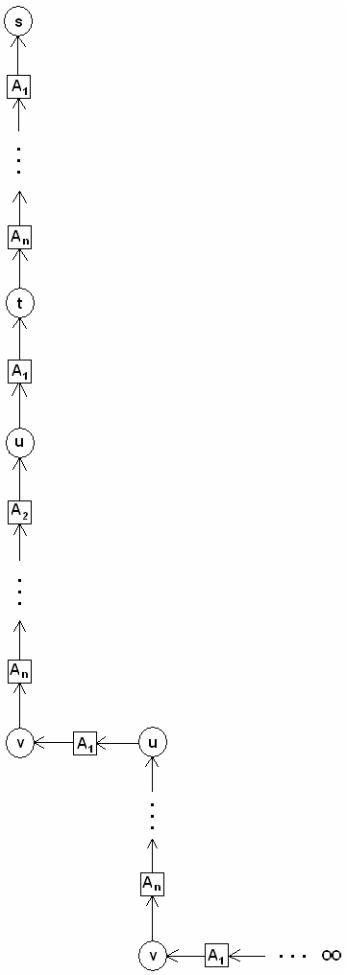
Kuva 9.9. Aligraafi GSS:ssä, jossa on sykli merkeillä $A_1 ... A_n$.

Aligraafin rakentamisessa ei ole suoritettu yhtään siirtoa, joten kaikki tilat graafissa on luotu samalla algoritmin askeleella *i*. Esitetään se siten kuvan 9.10 muodossa.



Kuva 9.10. Kuvan 9.9 aligraafi eri tavalla esitettynä.

Jos GSS:ää rakennetaan Tomitan algoritmin 2 mukaan, on jokainen tilasolmu graafissa eri joukossa $U_{i,j}$. Koska $A_1 \Rightarrow^* \varepsilon$, tilassa v on redusointi tyhjään sääntöön perustuen ja apulauseen 1 mukaan tilasta v on siirtymä tilaan u symbolilla A_1. Koska DFA sisältää syklin symboleilla $A_1 \ldots A_n$, päädytään GSS:ssä uudestaan tilaan v. Algoritmi ei siis pysähdy (kuva 9.11). □



Kuva 9.11. Päättymätön rekursio Tomitan algoritmissa 2.

10. Muunneltu Tomitan algoritmi 2

10.1. Muunneltu Tomitan algoritmi 2a

Vasemmanpuoleisen rekursion sisältävästä kieliopista muodostettu LR(1)-jäsennyksen mukainen DFA sisältää syklin. Tomitan algoritmissa luodaan aina uusi joukko $U_{i,j}$, kun suoritetaan tyhjään sääntöön perustuva redusointi. Jos kielioppi sisältää epäsuoran vasemman rekursion, päädytään kiertämään DFA:ssa olevaa sykliä luoden aina uusi alijoukko $U_{i,j}$ jokaista syklin tilaa kohti. Tämä prosessi ei pääty koskaan. Edellä huomattiin, että algoritmissa 1e vastaavasssa tilanteessa luotiin kahden askeleen mittainen polku kahden samassa joukossa U_i olevan tilasolmun v ja u välille, jos solmun v tilassa suoritettava redusointi osoittaa solmun u tilaan. Edellä huomattiin myös, että solmut tiloilla v ja u oli luotu tyhjään sääntöön perustuvien redusointien kautta, jolloin Tomitan algoritmissa 2 ne kuuluvat eri joukkoihin $U_{i,j}$. Niinpä tilasolmusta tilalla v luotiin uusi tilasolmu tilalla u, vaikka algoritmin askeleella i oli jo aikaisemmin luotu tilasolmu tilalla u.

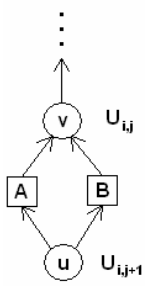
Tähän seikkaan perustuen tehdään Tomitan algoritmiin pieni muunnos. Muunnos koskee ainoastaan funktiota E-REDUSOIJA.

```
E-REDUSOIJA(i) {
  U_{i,j+1} = θ
  jokaiselle alkiolle (v, p) joukossa R_ε {
    olkoon X säännön p vasemman puolen symboli ja ja olkoon h solmun v tila.
           Olkoon gl merkintä LR(1)-jäsennystaulun kohdassa (h, X)
    jos joukossa U_{i,j+1} on tilasolmu [w, l]
      luo GSS:ään uusi symbolisolmu [u, X] ja tee solmusta u solmun w
           jälkeläinen ja solmun v edeltäjä
    muuten {
      luo GSS:ään tilasolmu [w, l] ja symbolisolmu [u, X]
      tee symbolisolmusta u solmun w seuraaja ja solmun v jälkeläinen
      lisää w joukkoon U_{i,j+1}
    }
    aseta R_ε = θ, A = U_{i,j+1}, j = j + 1
  }
}
```

Muunnos koskee algoritmin riviä

jos joukossa $U_{i,j+1}$ on tilasolmu $[w,\ l]$.

Tässä siis tarkistetaan vain, onko joukkoon $U_{i,j+1}$ jo lisätty funktion E-REDUSOIJA **jokaiselle**-silmukan aiemmalla kierroksella sellainen tilasolmu, jonka tilaa ollaan nyt lisäämässä tällä silmukan kierroksella (kuva 10.1).



Kuva 10.1. Funktion E-REDUSOIJA suorittama redusointi.

Muutetaan algoritmia niin, että tässä kohtaa tarkistetaan, onko missä tahansa joukossa $U_{i,j}$ samalla algoritmin askeleella i tilasolmu tilalla l. Funktio E-REDUSOIJA tulee siten seuraavanlaiseen muotoon.

```
E-REDUSOIJA(i) {
  U_{i,j+1} = θ
  jokaiselle alkiolle (v, p) joukossa R_ε {
    olkoon X säännön p vasemman puolen symboli ja ja olkoon h solmun v tila.
          Olkoon gl merkintä LR(1)-jäsennystaulun kohdassa (h, X)
    jos missä tahansa joukossa U_{i,k}, 0 ≤ k ≤ j+1, on tilasolmu [w, l]
      luo GSS:ään uusi symbolisolmu [u, X] ja tee solmusta u solmun w
          jälkeläinen ja solmun v edeltäjä
    muuten {
      luo GSS:ään tilasolmu [w, l] ja symbolisolmu [u, X]
      tee solmusta u solmun w seuraaja ja solmun v jälkeläinen
      lisää w joukkoon U_{i,j+1}
    }
    aseta R_ε = θ, A = U_{i,j+1}, j = j+1
  }
}
```

Mitään muuta muutosta Tomitan algoritmiin 2 ei tehdä. Tätä algoritmia kutsutaan *muunnelluksi Tomitan algoritmiksi 2a*.

10.2. Muunneltu Tomitan algoritmi 2a ja epäsuora vasen rekursio

Tarkastellaan epäsuoran vasemman rekursion sisältävää kielioppia Γ_6, jonka säännöt ovat

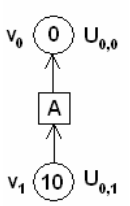
S' ::= S
S ::= ABSa | a
A ::= ε
B ::= ε.

LR(1)-jäsennystaulu on esitetty kuvassa 10.2. Syötemerkkijonona on *aa*.

	$	a	A	B	S
0		r3/s1	g10		g9
1		r1			
2		s1			
3		r3/s7	g6		g2
4	r1				
5		s4			
6		r4		g3	
7		r2			
8		r3/s7	g6		g5
9	hyv.				
10		r4		g8	
11	r2				

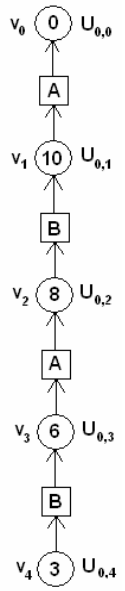
Kuva 10.2. LR(1)-jäsennystaulu kieliopille Γ_6.

Aluksi GSS:ään asetetaan tilasolmu [v_0, 0], ja tämä solmu asetetaan myös joukkoon $U_{0,0}$. LR(1)-jäsennystaulussa kohdassa (0, *a*) on toiminnot *r*3 ja *s*11. Joukkoon Q asetetaan (v_0, 11). Kieliopin Γ_3 sääntö 3 on $A ::= \varepsilon$, joten asetetaan joukkoon R_ε alkio (v_0, 3). Funktio E-REDUSOIJA käsittelee alkion (v_0, 3) joukosta R_ε ja luo tilasolmun [v_1, 10] ja symbolisolmun symbolilla A, josta tehdään solmun v_1 jälkeläinen ja solmun v_0 edeltäjä. Solmu v_1 asetetaan joukkoon $U_{0,1}$ (kuva 10.3).



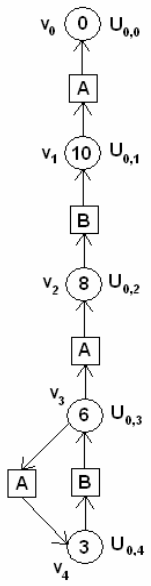
Kuva 10.3. Redusointi sääntöön $A ::= \varepsilon$ perustuen.

Siirtämättä yhtään syötemerkkiä päästään kuvan 10.4 tilanteeseen.



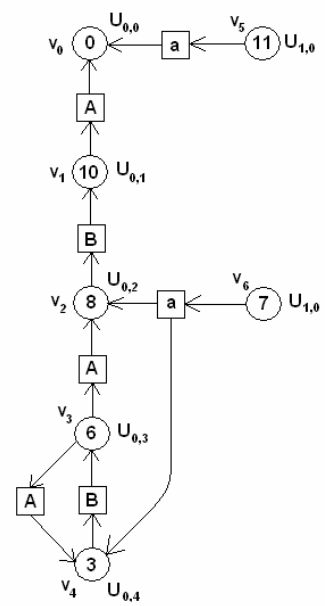
Kuva 10.4. Kolme redusointia perustuen ensin sääntöön $B ::= \varepsilon$, sitten sääntöön $A ::= \varepsilon$ ja viimeiseksi taas sääntöön $B ::= \varepsilon$.

Tilasolmuja v_2 ja v_4 käsiteltäessä joukkoon Q on lisätty alkiot (v_2, 7) ja (v_4, 7). Solmua v_4 käsiteltäessä on myös lisätty joukkoon R_ε alkio (v_4, 3). Funktio E-REDUSOIJA käsittelee alkion (v_0, 3) joukosta R_ε. LR(1)-jäsennystaulun kohdassa (3, *A*) on toiminta *g*6. Joukossa $U_{0,3}$ on kuitenkin tilasolmu [v_3, 6]. Niinpä luodaan GSS:ään symbolisolmu symbolilla *A* ja tehdään tästä solmusta solmun v_3 jälkeläinen ja solmun v_4 edeltäjä (luoden näin sykli symboleilla *A* ja *B*) (kuva 10.5).



Kuva 10.5. Redusointi sääntöön $A ::= \varepsilon$ perustuen.

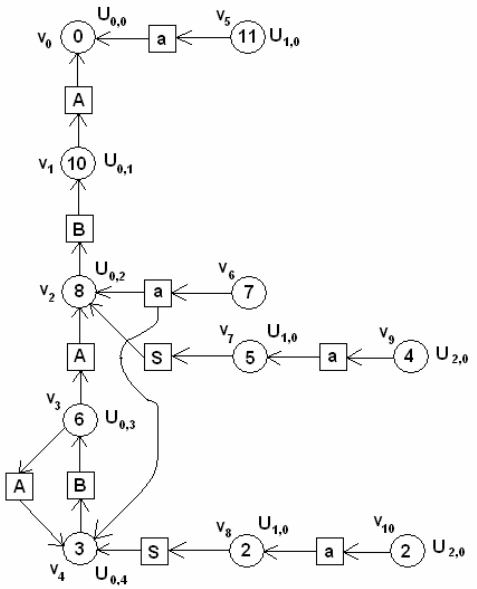
Suoritetaan seuraavaksi siirrot joukosta Q ja lisätään GSS:ään tilasolmut v_5 ja v_6, jotka asetetaan joukkoon $U_{1,0}$ (kuva 10.6).



Kuva 10.6. Syötemerkin *a* lukeminen.

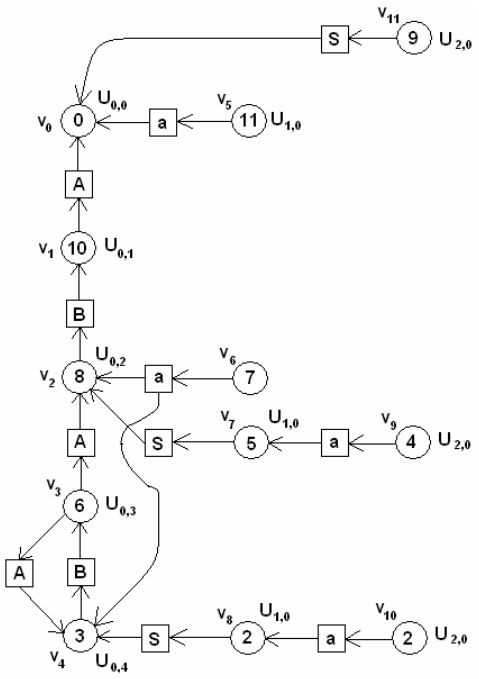
Käsiteltäessä solmua v_5 huomataan, ettei LR(1)-jäsennystaulussa ole toimintoja kohdassa (11, *a*). Solmun v_6 käsitteleminen vie takaisin solmuihin v_2

ja v_4. Solmussa v_2 luodaan tilasolmu [v_7, 5] sekä symbolisolmu symbolilla *S*, josta tehdään solmun v_7 jälkeläinen ja solmun v_2 edeltäjä. Solmu v_7 asetetaan joukkoon $U_{1,0}$. Solmussa v_4 luodaan tilasolmu [v_8, 2], sekä symbolisolmu symbolilla *S*, josta tehdään solmun v_8 jälkeläinen ja solmun v_4 edeltäjä. Solmun v_7 käsitteleminen lisää alkion (v_7, 4) ja solmun v_8 käsitteleminen alkion (v_8, 2) joukkoon *Q*. Käsitellään alkio (v_7, 4) joukosta *Q* ja luodaan tilasolmu [v_9, 4] sekä symbolisolmu symbolilla *a*, josta tehdään solmun v_9 jälkeläinen ja solmun v_7 edeltäjä. Solmu v_9 asetetaan joukkoon $U_{2,0}$. Seuraavaksi käsitellään joukon *Q* alkio (v_8, 2) ja luodaan tilasolmu [v_{10}, 2], sekä symbolisolmu symbolilla *a*, josta tehdään solmun v_{10} jälkeläinen ja solmun v_8 edeltäjä. Solmu v_{10} asetetaan joukkoon $U_{2,0}$ (kuva 10.7).



Kuva 10.7. Syötemerkin *a* lukeminen.

Solmun v_9 käsitteleminen vie takaisin solmuun v_0. Siellä luodaan solmu [v_{11}, 9] sekä symbolisolmu symbolilla *S*, josta tehdään solmun v_{11} jälkeläinen ja solmun v_0 edeltäjä. Solmu v_{11} asetetaan joukkoon $U_{2,0}$. Solmun v_{10} käsittelemisessä huomataan, ettei LR(1)-jäsennystaulussa ole toimintoa kohdassa (1, *$*), joten tässä yhteydessä ei tehdä mitään. Tila 9 on lopputila, joten syöte hyväksytään (kuva 10.8).



Kuva 10.8. Täydellinen GSS syötteelle *aa*.

10.3. Muunneltu Tomitan algoritmi 2a ja syklinen kielioppi

Tarkastellaan muunneltua Tomitan algoritmia 2 kieliopilla Γ_7, jonka säännöt ovat

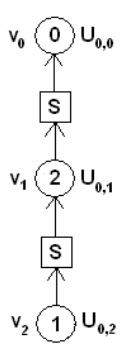
$S ::= SS$
$S ::= x$
$S ::= \varepsilon$.

Kielioppi on yksi Farshin [NF91] esittämistä kieliopeista, jotka aiheuttavat päättymättömän silmukan Tomitan algoritmissa 2. Kielioppi on *syklinen kielioppi* (cyclic grammar), koska se sisältää apumerkin S, joka jäsennyksessä tuottaa itsensä ($S \Rightarrow SS \Rightarrow S$). Kieliopista muodostettu LR(1)-jäsennystaulu on esitetty kuvassa 10.9. Syötemerkkijonona on x.

	$	x	S
0	r3	r3/s3	g2
1	r1/r3	r1/r3/s3	g1
2	hyv./r3	r3/s3	g1
3	r2	r2	

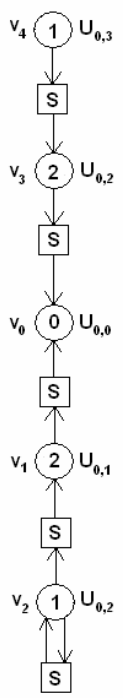
Kuva 10.9. LR(1)-jäsennystaulu kielopille Γ_7.

Siirtämättä yhtään syötemerkkiä päädytään kuvan 10.10 tilanteeseen.



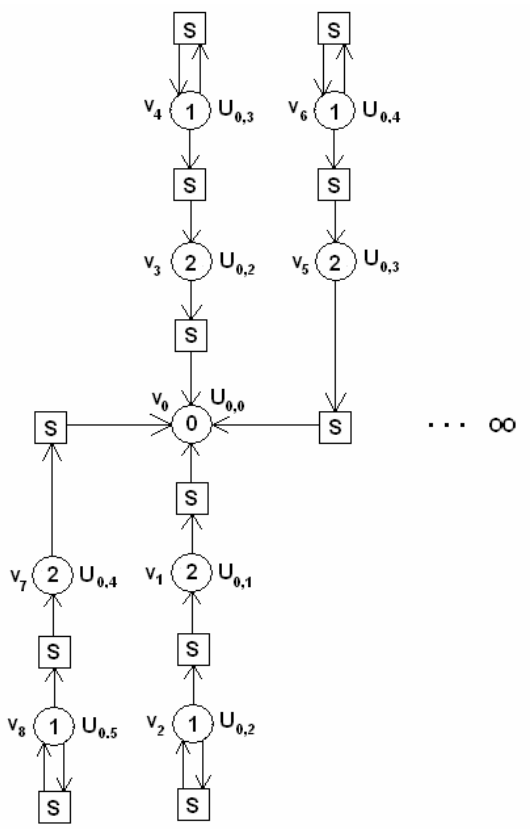
Kuva 10.10. Kaksi redusointia sääntöön $S ::= \varepsilon$ perustuen.

Käsitellessään tilasolmua v_2 funktio TOIMIJA lisää joukkoon R redusoinnin säännön 1 ($S ::= SS$) mukaan, sekä joukkoon R_ε redusoinnin säännön 3 ($S ::= \varepsilon$) mukaan. Suorittamalla redusointi joukosta R funktiossa REDUSOIJA, päädytään takaisin solmuun v_0. LR(1)-jäsennystaulun kohdassa (0, *S*) on toiminto *g2*. Nykyisessä joukossa $U_{0,2}$ ei ole tilasolmua tilalla 2, joten luodaan tilasolmu [v_3, 2], sekä symbolisolmu symbolilla *S*, josta tehdään solmun v_3 jälkeläinen ja solmun v_0 edeltäjä. Edelleen yhtään syötemerkkiä siirtämättä päädytään kuvan 10.11 tilanteeseen.



Kuva 10.11. Kolme redusointia perustuen ensin sääntöön $S ::= \varepsilon$, sitten sääntöön $S ::= SS$ ja viimeiseksi taas sääntöön $S ::= \varepsilon$.

Kuten edellä, käsitellessään solmua v_2, funktio TOIMIJA lisää joukkoon R redusoinnin säännön 1 mukaan, sekä joukkoon R_ε redusoinnin säännön 3 mukaan. Muunneltu Tomitan algoritmi 2a päätyy näin päättymättömään silmukkaan (kuva 10.12).



Kuva 10.12. Päättymätön silmukka muunnellussa Tomitan algoritmissa 2a.

Sääntö $S ::= SS$ aiheuttaa sen, että merkkijonolle x on ääretön määrä erilaisia jäsenyksiä. Yleisesti ottaen säännöt muotoa $A ::= A...A$ vievät muunnellun Tomitan algoritmin 2 päättymättömään silmukkaan.

Teoreema 2 *Jos kieliopissa on muotoa* $A ::= A...A$, *missä* $A \overset{+}{\Rightarrow} \varepsilon$, *oleva sääntö, muunneltu Tomitan algoritmi 2a joutuu päättymättömään silmukkaan käsitellessään sellaista LR(1)-jäsennyksen mukaisen DFA:n tilaa, joka sisältää LR(1)-alkion muotoa* $[A ::= \bullet A \ ...A, x]$.

Todistus Oletetaan, että nykyinen alkiojoukko on $U_{i,j}$ ja että käsitellään tilaa h tilasolmussa v. Jos tilassa h on muotoa $[A ::= \bullet A \ ...A, x]$, missä $A \overset{+}{\Rightarrow} \varepsilon$, oleva LR(1)-alkio, LR(1)-alkiojoukkoja muodostavan algoritmin sulkeumaoperaatio takaa sen, että tilassa h on myös muotoa $[B ::= \bullet, x]$, missä $A \overset{*}{\Rightarrow} B$, ($A \overset{+}{\Rightarrow} \varepsilon$, eli FIRST($Ax$) = x) oleva LR(1)-alkio. Tämän redusoinnin suorittaminen luo GSS:ään tilasolmun, joka asetetaan joukkoon $U_{i,j+1}$. LR(1)-jäsennyksen mukainen DFA takaa sen, että tilasta h on siirtymä symbolilla A tilaan k, jossa

on LR(1)-alkio [$A ::= A \bullet ...A, x$]. GSS:ää rakentaessa siihen lisätään tilasolmu [w, k], josta kahden askeleen mittainen polku tilasolmuun v. Yhtään syötemerkkiä siirtämättä päädytään lisäämään GSS:ään tilasolmu [z, l], jossa DFA:ssa on LR(1)-alkio [$A ::= A ...A \bullet, x$]. Solmu z asetetaan joukkoon $U_{i,j+k}$, missä k on säännön $A ::= A...A$ oikean puolen pituus. Suorittamalla redusointi säännön $A ::= A...A$ perusteella tilassa l, kuljetaan takaisin solmuun v. Koska tämän redusoinnin suorittaa funktio REDUSOIJA lisätään joukkoon $U_{i,j+k}$ tilasolmu [t, k], vaikka tällä algoritmin viimeisimmällä askeleella on jo tilasolmu samalla tilalla lisätty GSS.ään. Niinpä GSS:n rakentaminen jatkuu samalla tavalla kuin tilasolmusta w eteenpäin tehtiin, ja päädytään loppumattomaan prosessiin. □

10.4. Muunneltu Tomitan algoritmi 2b

Edellä huomattiin muunnellun Tomitan algoritmin 2a päätyvän päättymättömään silmukkaan, kun kielioppi sisälsi muotoa $A ::= A...A$ olevan säännön. Tällöin funktiossa REDUSOIJA lisättiin sellainen tilasolmu joukkoon $U_{i,k}$, että askeleella i oli jo sama tilasolmu lisätty joukkoon $U_{i,j}$, $j < k$. Tämä algoritmin ominaisuus estää tilanteet, joissa hylätään kieliopin määräämään kieleen kuuluva lause. Edellä on huomattu, että tällaisissa tapauksissa kielioppi sisältää epäsuoran oikean rekursion. Kieliopissa, joka sisältää muotoa $A ::= A...A$ olevan säännön, ongelmana on kuitenkin se, että luotavasta tilasolmusta on kahden askeleen pituinen polku tilasolmuun, joka on luotu samalla algoritmin askeleella i.

Muunnellussa Tomitan algoritmissa 2a funktio REDUSOIJA oli seuraavanlainen:

```
REDUSOIJA(i) {
  poista (u, p) joukosta R
  olkoon m säännön p oikean puolen pituus ja olkoon X
       säännön p vasemman puolen symboli
  jokaiselle tilasolmulle w joka voidaan saavuttaa solmusta u
       (2m-1) pituista polkua pitkin suorita {
    olkoon k solmun w tila ja olkoon gl toiminta LR(1)-jäsennystaulun
            kohdassa (k, X)
    jos joukossa U_{i,j} ei ole tilasolmua tilalla l, luo uusi tilasolmu v tilalla l
            ja lisää v joukkoihin U_{i,j} ja A
    olkoon v tilasolmu joukosta U_{i,j} tilalla l
    jos GSS:ssä on olemassa kahden askeleen mittainen polku solmusta v
      solmuun w, älä tee mitään
    muuten {
      luo GSS:ään uusi symbolisolmu u' symbolilla X
```

```
      tee symbolisolmusta u' solmun v seuraaja ja solmun w jälkeläinen
      jos v ei ole joukossa A {
        jokaiselle redusoinnille rq LR(1)-jäsennystaulun kohdassa (l, a_{i+1}), missä q ei
                    ole tyhjä sääntö
          lisää (u', q) joukkoon R
      }
    }
  }
}
```

Edellä kuvatussa ongelmasta muunnellun Tomitan algoritmi 2a päättymättömään silmukkaan vierässä tilanteessa huomio kiinnittyy funktion REDUSOIJA kohtaan:

```
      jokaiselle tilasolmulle w joka voidaan saavuttaa solmusta u
            (2m-1) pituista polkua pitkin suorita {
          olkoon k solmun w tila ja olkoon gl toiminta LR(1)-jäsennystaulun
                        kohdassa (k, X)
          jos joukossa U_{i,j} ei ole tilasolmua tilalla l, luo uusi tilasolmu v tilalla l
                    ja v joukkoihin U_{i,j} ja A.
```

Muotoa $A ::= A...A$ olevien sääntöjen takia tässä kohdassa on tarkistettava myös, ettei solmua w ole luotu samalla algoritmin askeleella i. Jos on, luodaan kahden askeleen mittainen polku tilasolmusta [v, l] solmuun w. Funktio REDUSOIJA tulee siten seuraavanlaiseen muotoon:

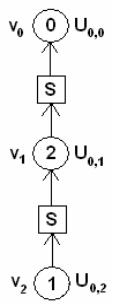
```
REDUSOIJA(i) {
  poista (u, j) joukosta R
  olkoon m säännön p oikean puolen pituus ja olkoon X
        säännön p vasemman puolen symboli
  jokaiselle tilasolmulle w joka voidaan saavuttaa solmusta u
        (2m-1) pituista polkua pitkin suorita {
    olkoon k solmun w tila ja olkoon gl toiminta LR(1)-jäsennystaulun
              kohdassa (k, X)
    jos joukossa U_{i,j} ei ole tilasolmua tilalla l ja w kuuluu joukkoon U_{g,h} missä g < i,
      luo uusi tilasolmu [v, l] ja lisää v joukkoihin U_{i,j} ja A
    jos askeleella i luoduissa tilasolmuissa ei ole solmua v tilalla l, luo tälläinen solmu
              ja aseta se joukkoihin U_{i,j} ja A
    olkoon [v, l] sellaisen joukon U_{i,h} tilasolmu, että k on on mahdollisimman suuri
              luku
    jos GSS:ssä on olemassa kahden askeleen mittainen polku solmusta v
      solmuun w, älä tee mitään
    muuten {
      luo GSS:ään uusi symbolisolmu [u', X]
      tee solmusta u' solmun v seuraaja ja solmun w jälkeläinen
```

```
        jos v ei ole joukossa A {
          jokaiselle redusoinnille rq LR(1)-jäsennystaulun kohdassa (l, a_{i+1}), missä q ei
                    ole tyhjä sääntö
            lisää (u', q) joukkoon R

        }
      }
    }
  }
```

Mitään muuta muutosta muuneltuun Tomitan algoritmiin 2a ei tehdä. Tätä algoritmia kutsutaan *muunnelluksi Tomitan algoritmiksi 2b*.

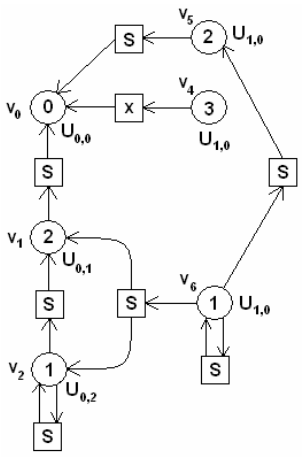
10.5. Esimerkki muunnellusta Tomitan algoritmilla 2b

Tarkastellaan muunnellun Tomitan algoritmin 2b toimintaa kieliopilla Γ_7 luvusta 10.4 syötemerkkijonolla *x*. Siirtämättä yhtään syötemerkkiä päästään kuvan 10.13 tilanteeseen.



Kuva 10.13 Kaksi redusointia sääntöön $S ::= \varepsilon$ perustuen.

Käsitellessään tilasolmua v_2 funktio TOIMIJA lisää joukkoon *R* redusoinnin säännön 1 ($S ::= SS$) mukaan, sekä joukkoon R_ε redusoinnin säännön 3 ($S ::= \varepsilon$) mukaan. Suorittamalla redusointi joukosta *R* funktiossa REDUSOIJA, päädytään takaisin solmuun v_0. LR(1)-jäsennystaulun kohdassa (0, *S*) on toiminto *g*2. Nyt funktiossa REDUSOIJA huomataan, että solmu v_0 on luotu nykyisellä algoritmin askeleella *i*, joten ei luoda uutta tilasolmua tilalla 2. Solmusta v_0 on jo kahden askeleen mittainen polku tilasolmuun [v_1, 1], joten ei tehdä mitään. Algoritmin suoritusta jatkamalla päädytään lopulta kuvan 10.14 tilanteeseen.



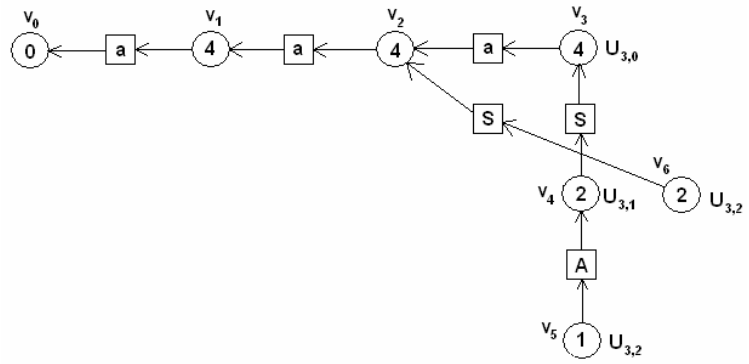
Kuva 10.14. Täydellinen GSS syötteelle x.

Tila 2 solmussa v_5 on lopputila, joten syöte hyväksytään.

10.6. Muunneltu Tomitan algoritmi 2b ja epäsuora oikea rekursio

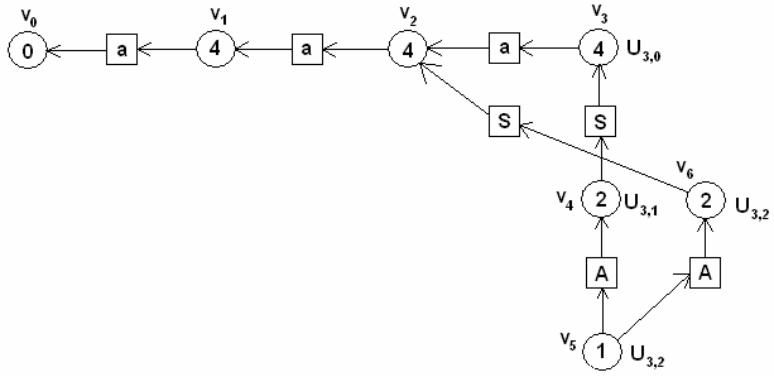
Algoritmiin on tehtävä vielä yksi muutos. Muutos, joka tehtiin muunneltuun Tomitan algoritmiin 2a, koski funktiota E-REDUSOIJA. Funktiossa tarkastettiin, sisältääkö mikään algoritmin askeleella i lisätty tilasolmu sellaista tilaa, johon E-REDUSOIJAN käsittelemä redusointi osoittaa. Jos tämän tilan omaava tilasolmu oli jo luotu, algoritmin nykyisellä askeleella i siihen luotiin kahden askeleen mittainen polku, jos tällaista polkua ei vielä ollut olemassa. Edellä huomattiin, että algoritmi pystyi tällöin tunnistamaan oikein syötteen käsitellessään kielioppia, joka sisälsi epäsuoran vasemmanpuoleisen rekursion. Alkuperäisessä Tomitan algoritmissa 2 näissä tilanteissa algoritmi joutui päättymättömään silmukkaan. Nyt kuitenkin kieliopeilla, jotka sisältävät epäsuoran oikean rekursion, muunneltu Tomitan algoritmi 2b päätyy hylkäämään syötteen, joka kuuluu kieliopin määräämään kieleen.

Tarkastellaan jälleen luvun 8 kielioppia Γ_5 ja syötettä *aaa*. Kuvan 10.15 tilanteessa on lisätty tila 2 toiseen kertaan algoritmin askeleella 3, kun GSS:ään on lisätty tilasolmu v_6.



Kuva 10.15. Redusointi sääntöön $S ::= aSA$ perustuen.

Seuraavaksi suoritetaan redusointi solmusta v_6. Redusoinnin suorittaa funktio E-REDUSOIJA. Se tarkistaa, onko tällä algoritmin askeleella luotu tilasolmua tilalla 1. Koska v_5 on tälläinen tilasolmu, luodaan solmusta v_6 solmuun v_5 kahden askeleen mittainen polku. Muita toimintoja ei enää algoritmissa tehdä, joten algoritmi pysähtyy. Lopputilaa 3 ei ole algoritmin viimeisellä askeleella *i* luoduissa tilasolmuissa, joten syöte *aaa* hylätään (kuva 10.16).



Kuva 10.16. Täydellinen muunnellun Tomitan algoritmin 2b muodostama GSS syötteelle *aaa*.

Tarkasteltaessa muunneltua Tomitan algoritmia 2b huomataan, että nykyisellä algoritmin askeleella *i* siinä tapauksessa, että $j > 0$, joukoista $U_{i,k}$, missä $k < j$, merkin *k* arvolla ei ole merkitystä. Tarkemmin sanottuna, funktiossa REDUSOIJA kaikki joukkojen algoritmin askeleella *i* luodut solmut otetaan huomioon ja funktiossa E-REDUSOIJA ei merkin *k* arvolla sinänsä ole merkitystä, koska joka tapauksessa $0 \le k < j+1$. Näin ollen tehdään funktioon

REDUSOIJA vielä sellainen muutos, että kuljettaessa redusoinnissa sellaiseen solmun, joka on luotu algoritmin jollain aikaisemmalla askeleella, asetetaan kaikki nykyisellä algoritmin askeleella luodut solmut joukkoon $U_{i,-1}$, jos $j > 0$. Funktio REDUSOIJA tulee siten seuraavaan muotoon:

```
REDUSOIJA(i) {
  poista (u, j) joukosta R
  olkoon m säännön p oikean puolen pituus ja olkoon X
       säännön p vasemman puolen symboli
  jokaiselle tilasolmulle w joka voidaan saavuttaa solmusta u
            (2m-1) pituista polkua pitkin suorita {
    olkoon k solmun w tila ja olkoon gl toiminta LR(1)-jäsennystaulun
            kohdassa (k, X)
    jos joukossa U_{i,j} ei ole tilasolmua tilalla l ja w kuuluu joukkoon U_{g,h}, missä g < i {
      jos j > 0 {
        aseta kaikki joukkojen U_{i,g}, missä g ≤ j, solmut joukkoon U_{i,-1}
        j = 0
      }
      luo uusi tilasolmu [v, l] ja lisää v joukkoihin U_{i,j} ja A
    }
    jos askeleella i luodusta tilasolmuissa ei ole solmua [v, l], luo tälläinen solmu ja
            aseta se joukkoihin U_{i,j} ja A
    olkoon [v, l] sellaisen joukon U_{i,h} tilasolmu, missä luku k on on mahdollisimman
            suuri
    jos GSS:ssä on olemassa kahden askeleen mittainen polku solmusta v
            solmuun w, älä tee mitään
    muuten {
      luo GSS:ään uusi symbolisolmu [u', X]
      tee symbolisolmusta u' solmun v seuraaja ja solmun w jälkeläinen
      jos v ei ole joukossa A {
        jokaiselle redusoinnille rq LR(1)-jäsennystaulun kohdassa (l, a_{i+1}), missä q ei
                ole tyhjä sääntö
          lisää (u', q) joukkoon R
      }
    }
  }
}
```

Muita muutoksia ei muunneltuun Tomitan algoritmiin 2b ei tehdä. Kutsutaan tätä algoritmia *muunnelluksi Tomitan algoritmiksi 2c*.

10.7. Muunneltu Tomitan algoritmi 2c

Muunneltu Tomitan algoritmi 2c on kokonaisuudessaan seuraavanlainen.

Algoritmi 7. Muunneltu Tomitan yleistetty LR-jäsennysalgoritmi kontekstittomille kieliopeille.

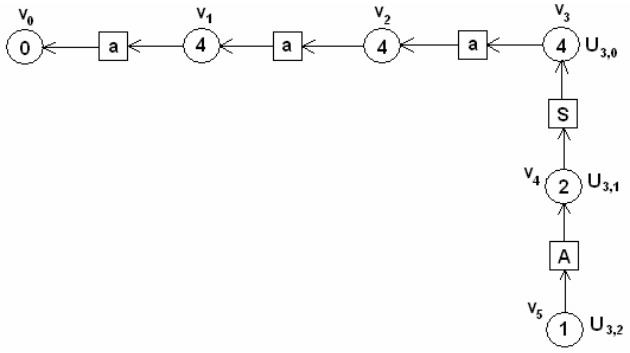
luo tilasolmu [v_0 , 0], missä 0 on DFA:n alkutila
aseta $U_{0,0} = \{v_0\}$, $A = \theta$, $R = \theta$, $R_\varepsilon = \theta$, $Q = \theta$
jokaiselle i, $0 \le i \le n$, **suorita** JÄSENNÄ_SYMBOLI(i)
olkoon q DFA:n lopputila
jos $U_{n,i}$, $0 \le i \le j$ sisältää tilasolmun, jonka tila on q, ilmoita jäsennyksen onnistuneen
muuten ilmoita jäsennyksen epäonnistuneen

```
JÄSENNÄ_SYMBOLI(i) {
  A = U_{i,0}
  U_{i+1,0} = θ
  niin kauan kuin A ≠ θ tai R ≠ θ tai R_ε ≠ θ
    jos A ≠ θ suorita TOIMIJA(i)
    muuten jos R ≠ θ suorita REDUSOIJA(i)
    muuten jos R_ε ≠ θ suorita E-REDUSOIJA(i)
}

TOIMIJA(i) {
  poista solmu v joukosta A ja olkoon h solmun v tila
  jos 'siirrä k' on toiminta LR(1)-jäsennystaulun kohdassa (h, a_{i+1})
    lisää (v, k) joukkoon Q
  jokaiselle 'redusoi p' toiminnolle LR(1)-jäsennystaulun kohdassa (h, a_{i+1}),
            misssä säännön p oikean puolen pituus on nollaa suurempi
    jokaiselle solmun v jälkeläissolmulle u, lisää (u, p) joukkoon R
  jokaiselle 'redusoi p' toiminnolle LR(1)-jäsennystaulun kohdassa (h, a_{i+1}),
            missä säännön p oikean puolen pituus on nolla
      lisää (v, p) joukkoon R_ε
}

REDUSOIJA(i) {
  poista (u, j) joukosta R
  olkoon m säännön p oikean puolen pituus ja olkoon X
       säännön p vasemman puolen symboli
  jokaiselle tilasolmulle w joka voidaan saavuttaa solmusta u
            (2m-1) pituista polkua pitkin suorita {
    olkoon k solmun w tila ja olkoon gl toiminta LR(1)-jäsennystaulun
            kohdassa (k, X)
    jos joukossa U_{i,j} ei ole tilasolmua tilalla l ja w kuuluu joukkoon U_{g,h}, missä g < i {
      jos j > 0 {
        aseta kaikki joukkojen U_{i,g}, missä g ≤ j, solmut joukkoon U_{i,-1}
        j = 0
      }
```

```
      luo uusi tilasolmu [v, l] ja lisää v joukkoihin U_{i,j} ja A
    }
    jos askeleella i luodusta tilasolmuissa ei ole solmua [v, l], luo tällainen solmu ja
        aseta se joukkoihin U_{i,j} ja A
    olkoon [v, l] sellaisen joukon U_{i,h} tilasolmu, missä luku k on on mahdollisimman
        suuri
    jos GSS:ssä on olemassa kahden askeleen mittainen polku solmusta v
        solmuun w, älä tee mitään
    muuten {
      luo GSS:ään uusi symbolisolmu [u', X]
      tee symbolisolmusta u' solmun v seuraaja ja solmun w jälkeläinen
      jos v ei ole joukossa A {
        jokaiselle redusoinnille rq LR(1)-jäsennystaulun kohdassa (l, a_{i+1}), missä q ei
            ole tyhjä sääntö
          lisää (u', q) joukkoon R
      }
    }
  }
}

E-REDUSOIJA(i) {
  U_{i,j+1} = θ
  jokaiselle alkiolle (v, p) joukossa R_ε {
    olkoon X säännön p vasemman puolen symboli ja ja olkoon h solmun v tila.
        Olkoon gl merkintä LR(1)-jäsennystaulun kohdassa (h, X)
    jos missä tahansa joukossa U_{i,k}, 0 ≤ k ≤ j+1, on tilasolmu [w, l]
      luo GSS:ään uusi symbolisolmu [u, X] ja tee solmusta u solmun w
          jälkeläinen ja solmun v edeltäjä
    muuten {
      luo GSS:ään tilasolmu [w, l] ja symbolisolmu [u, X]
      tee solmusta u solmun w seuraaja ja solmun v jälkeläinen
      lisää w joukkoon U_{i,j+1}
    }
    aseta R_ε = θ, A = U_{i,j+1}, j = j+1
  }
}

SIIRTÄJÄ(i) {
  niin kauan kuin Q ≠ θ suorita {
    poista elementti (v, k) joukosta Q
    jos joukossa U_{i+1,0} ei ole tilasolmua [w, k], luo tällainen solmu
    jos solmulla w ei ole jälkeläissolmua [u, a_{i+1}], luo tällainen solmu
    jos solmu u ei ole solmun v edeltäjä, tee solmusta u solmun v edeltäjä
  }
}
```

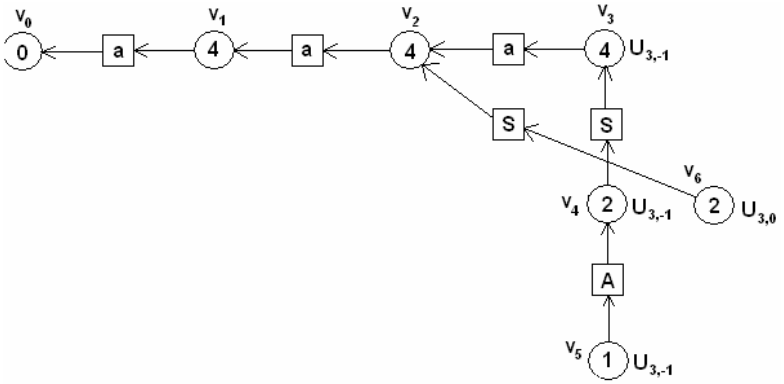
10.8. Muunneltu Tomitan algoritmi 2c ja epäsuora oikea rekursio

Tarkastellaan jälleen kielioppia Γ_5 ja syötettä *aaa*. Ensimmäisen kerran funktiota REDUSOIJA käytetään kuvan 10.17 tilanteessa, jossa suoritetaan redusointi solmusta v_5.



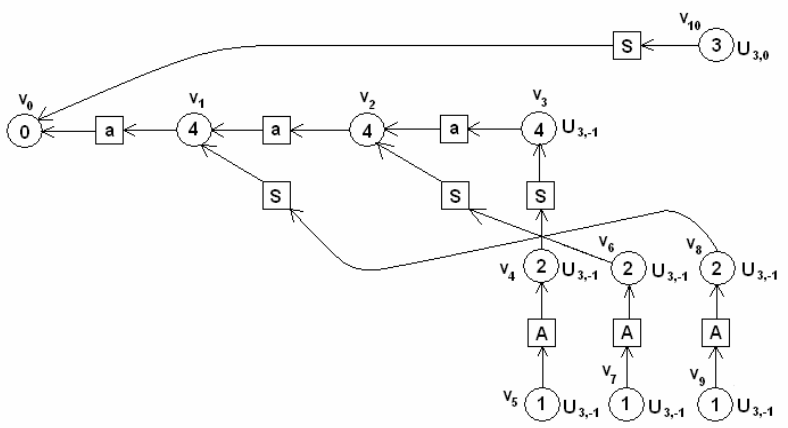
Kuva 10.17. Tilanne ennen funktion REDUSOIJA suorittamista.

Redusointi vie solmuun v_2, jota ei ole luotu algoritmin nykyisellä askeleella *i*, vaan se on luotu aiemmin. Nyt $j > 0$, joten kaikki algoritmin askelleella 3 luodut tilasolmut asetetaan joukoon $U_{3,-1}$. Tämän jälkeen luodaan uusi tilasolmu [v_6, 2], joka asetetaan joukkoon $U_{3,0}$ (kuva 10.18).



Kuva 10.18 Redusointi sääntöön $S ::= aSA$ perustuen.

Algoritmin suoritusta jatkamalla päästään lopulta kuvan 10.19 GSS:ään. Solmun v_{10} tila 3 on lopputila, joten syöte hyväksytään.



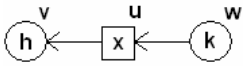
Kuva 10.19. Täydellinen GSS syötteelle *aaa*.

11. Muunnellun Tomitan algoritmin 2 ominaisuuksista

Esitellään seuraavaksi joitakin tuloksia muunneltuun Tomitan algoritmiin 2 liittyen.

11.1. Muunneltu Tomitan algoritmi 2c pysähtyy aina

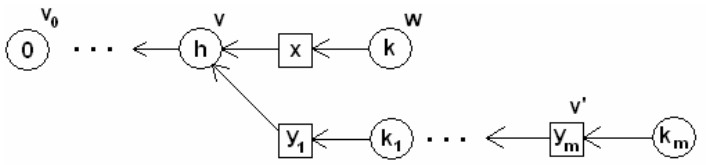
Seuraavaksi osoitetaan, että muunneltu Tomitan algoritmi 2c pysähtyy kaikilla syötteillä. Ensin todistetaan apulause, joka liittyy kahden askeleen mittaisiin polkuihin kahden sellaisen tilasolmun välissä, jotka on luotu samalla algoritmin askeleella *i*. Todistus perustuu Scottin ja muiden [SJH00] tekemään huomioon RNGLR-jäsentäjässä sellaisista samalla algoritmin askeleella *i* luoduista tilasolmuista, joiden välillä on kahden askeleen mittainen polku.



Kuva 11.1. Kahden askeleen mittainen polku solmusta *w* solmuun *v*.

Apulause 2 Jos muunnellun Tomitan algoritmin 2c mukaan rakennetussa GSS:ssä on kuvan 11.1 esittämä polku, jossa solmut *w* ja *v* on luotu samalla algoritmin askeleella *i*, niin $x \overset{*}{\Rightarrow} \varepsilon$. Lisäksi tila *h* sisältään LR(1)-jäsennyksen mukaisessa DFA:ssa LR(1)-alkion ($[x ::= \bullet\gamma],\ a_{i+1}$), missä $\gamma \overset{*}{\Rightarrow} \varepsilon$.

Todistus Koska solmut *v* ja *w* on luotu samalla algoritmin askeleella *i*, kaari (*u*, *v*) on täytynyt luoda suoritettaessa redusointia (*v'*, *p*). (Jos solmu *w* olisi luotu suorittamalla siirto solmussa *v*, solmuja *w* ja *v* ei olisi luotu samalla algoritmin askeleella *i*.) Symboli *x* on tällöin säännön *p* vasen puoli. Jos säännön *p* pituus on *m*, solmusta *v'* on 2*m*-1 pituinen polku solmuun *v* tai *v* = *v'*, jos *m* = 0. LR(1)-jäsennystaulun kohdassa (*h*, *x*) on merkintä *gk* (kuva 11.2).

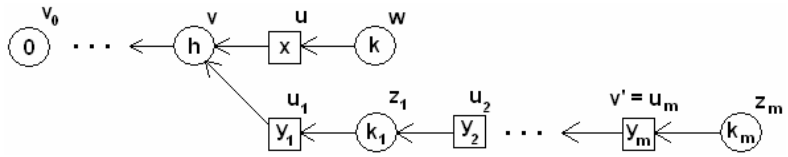


Kuva 11.2. (2*m*-1)-pituinen polku solmusta *v*′ solmuun *v*.

Muunnellun Tomitan algoritmin 2c ominaisuuksien perusteella tässä algoritmin suoritusvaiheessa on määrätty redusointi (*v*′, *p*) jos ja vain jos toinen seuraavista ehdoista on voimassa.

1. LR(1)-jäsennystaulun kohdassa (k_m, a_{i+1}) on merkintä *rp* ja sääntö *p* on tyhjä sääntö. Tällöin $m = 0$, $v = v'$, $k_m = h$. Redusointi (*v*′, *p*) on tällöin asetettu joukkoon R_ε.

2. On olemassa sellainen joukon $U_{i,j}$ tilasolmu $[z_m, k_m]$, että solmusta z_m on yhden askeleen mittainen polku solmuun *v*′ ja LR(1)-jäsennystaulun kohdassa (k_m, a_{i+1}) on merkintä *rp*, missä sääntö *p* on muotoa $x ::= y_1 ... y_m$ (kuva 11.3). Redusointi (*v*′, *p*) on tällöin asetettu joukkoon *R*.



Kuva 11.3. Redusointi sääntöön $x ::= y_1 ... y_m$ perustuen.

Koska solmu *v* on luotu nykyisellä algoritmin askeleella *i*, kaikki solmut z_d, $1 \le d \le m$, on luotu nykyisellä algoritmin askeleella *i*.

Todistetaan tulos induktiolla perustuen solmujen luontijärjestykseen GSS:ssä.

Jos (*w*, *u*) ja (*u*, *v*) ovat ensimmäisiä GSS:ään luotavia kaaria, on ainoa mahdollisuus se, että $x \Rightarrow \varepsilon$, jolloin $v = v_0$ ja $w \in U_{0,1}$. Tämä toteuttaa yllä esitetyn vaihtoehdon 1. Tila 0 sisältää tällöin LR(1)-alkion $([x ::= \bullet\varepsilon],\ a_1)$ $(x \Rightarrow \gamma = \varepsilon)$.

Oletetaan nyt, että tulos on on tosi kaarille, jotka luotiin ennen kaarta (*u*, *v*).

Jos (*v*′, *p*) on ylläolevista vaihtoehdoista vaihtoehdon 1 mukainen, niin tilassa *h* on muotoa $([x ::= \bullet\varepsilon],\ a_{i+1})$ oleva LR(1)-alkio. Kieliopissa on johto

$x \Rightarrow \gamma = \varepsilon$, eli tila *h* sisältää LR(1)-alkion muotoa ([$x ::= \bullet\gamma$], a_{i+1}), missä $\gamma \overset{*}{\Rightarrow} \varepsilon$, kuten vaadittiin.

Jos redusointi (*v*′, *p*) on ylläolevista vaihtoehdoista vaihtoehdon 2 mukainen, niin jokainen kaari (u_d, z_{d-1}), missä $1 \le d \le m$ ja $z_0 = v$, on luotu ennen kaarta (*u*, *v*). Silloin induktion mukaan $y_d \overset{*}{\Rightarrow} \varepsilon$, $1 \le d \le m$. Siis $x \Rightarrow y_1 ... y_m \overset{*}{\Rightarrow} \varepsilon$, ja koska LR(1)-jäsennystaulun kohta (k_m, a_{i+1}) sisältää merkinnän *rp*, DFA:n tilan k_m on sisällettävä LR(1)-alkio ([$x ::= y_1 ... y_m \bullet$], a_{i+1}). Tällöin jokainen tila k_{d-1}, $1 \le d \le m$, sisältää LR(1)-alkion ([$x ::= y_1 ... y_{d-1} \bullet y_d ... y_m$], a_{i+1}). Nyt siis *h* = k_0 ja tämä tila sisältää LR(1)-alkion ([$x ::= \bullet y_1 ... y_m$], a_{i+1}), missä $y_1 ... y_m \overset{*}{\Rightarrow} \varepsilon$, kuten vaadittiin. □

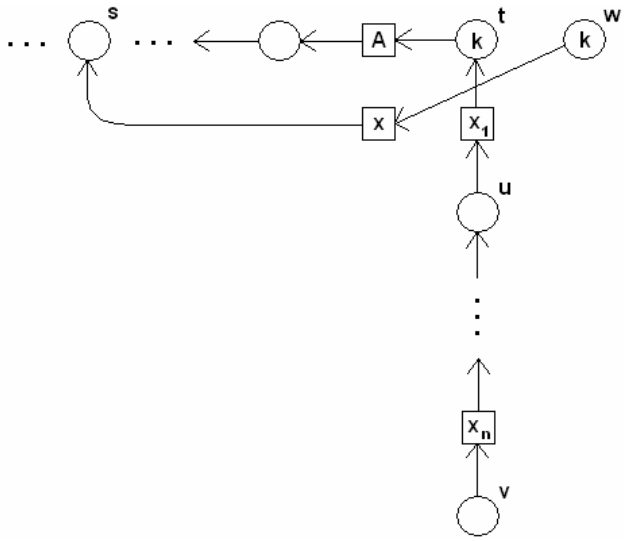
Seuraavaksi osoitetaan, että muunneltu Tomitan algoritmi 2c pysähtyy kaikilla syötteillä. Käydään ensin kaksi apulausetta läpi.

Apulause 3 *Muunnellun Tomitan algoritmin 2c askeleella* i *sellainen tila* s, *joka on lisätty GSS:ään tällä algoritmin askeleella, voidaan lisätä toiseen kertaan vain, jos on suoritettu redusointi, joka perustuu epäsuoran oikean rekursion sisältävään sääntöön.*

Todistus Funktiossa E-REDUSOIJA tarkistetaan, onko joukoissa $U_{i,k}$, $0 \le k \le j+1$ sellaista tilaa, joka on LR(1)-jäsennystaulun kohdassa (*h*, *X*), missä h on sen tilasolmun tila, josta redusointia suoritetaan ja X on redusoinnissa käytettävän (tyhjän) säännön vasen puoli. Jos tällaisen tilan omaava solmu on jo algoritmin askeleella i luotu, siihen luodaan kahden askeleen mittainen polku käsiteltävästä solmusta, ellei tällaista polkua ole jo olemassa. Tilanne jossa funktiossa E-REDUSOIJA luodaan sellainen tilasolmu, joka nykyisellä algoritmin askeleella on jo luotu on siis mahdollinen vain, kun nykyisen algoritmin askeleella luotuja tilasolmuja on siirretty joukkoon $U_{i,-1}$. Tarkastelemalla muunneltua Tomitan algoritmia 2c huomataan, että tämä on mahdollista vain, jos funktiossa REDUSOIJA kuljetaan sellaiseen tilaolmuun, jota ei ole luotu nykyisellä algoritmin askeleella *i* ja *j* > 0. Tällöin on siis nykyisellä algoritmin askeleella *i* täytynyt ennen funktion REDUSOIJA suorittamista suorittaa funktio E-REDUSOIJA ainakin kertaalleen.

Funktiossa REDUSOIJA tarkistetaan, sisältääkö nykyinen joukko $U_{i,j}$ jo solmun, jonka tila on LR(1)-jäsennystaulun kohdassa (*k*, *X*), missä k on sen solmun tila, johon on redusoinnin yhteydessä kuljettu ja X on redusoinnissa käytettävän säännön vasen puoli. Jos tällainen solmu on olemassa siihen luodaan kahden askeleen mittainen polku, mikäli tällaistä polkua ei ole jo olemassa. Siis mikäli funktiossa REDUSOIJA luodaan sellainen tilasolmu, että

saman tilan omaava solmu on jo luotu nykyisellä algoritmin askeleella *i*, on tällä askeleella luotu solmuja funktion E-REDUSOIJA kautta. Tämä funktio nimittäin muuttaa nykyisen joukon joukosta $U_{i,j}$ joukkoon $U_{i,j+1}$. Jos tilasolmu, johon redusoinnissa kuljetaan on luotu askeleella *i*, luodaan tästä solmusta kahden askeleen mittainen polku siihen solmuun, jonka tila on LR(1)-jäsennystaulun kohdassa (*k*, *X*). Jos askeella *i* ei tällaista solmua ole luotu, se luodaan joukkoon $U_{i,j}$. (Tämä on se muutos, joka tehtiin muunneltuun Tomitan algoritmiin 2b.). Niinpä redusoinnissa, jossa askeleella *i* luodaan toistamiseen tilasolmu tilalla, joka askeleella *i* jo on, on redusoinnissa kuljettu sellaiseen solmuun, joka on luotu aikaisemmalla algoritmin askeleella.



Kuva 11.4. Kaksi samalla algoritmin askeleella lisättyä tilasolmua samalla tilalla.

Viitataan kuvaan 11.4, jossa on esitetty edellä kuvattu tilanne yleisessä tapauksessa. Solmut *t*, *u*, *v* ja *w* on luotu samalla algoritmin askeleella *i*. Solmut solmusta *u* solmuun *v* on täytynyt luoda solmun *t* luonnin jälkeen ja kaikista niistä on nyt kahden askeleen pituinen polku solmuun, joka on luotu samalla algoritmin askeleella *i*. Apulauseen 2 mukaan tällöin $x_i \overset{*}{\Rightarrow} \varepsilon$, $1 \le i \le n$. Algoritmin askeleella *i* on myös lisätty tilasolmu *w*, jonka tila *k* on jo aikeisemmin tällä algoritmin askeleella luodulla tilasolmulla *t*. Solmu *w* on täytynyt luoda suorittamalla redusointi solmusta *v*. Tämän redusoinnin täytyy perustua muotoa $x ::= ...Ax_1...x_n$ olevaan sääntöön. LR(1)-jäsennyksen mukaisen DFA:n ominaisuuksien perusteella tiettyyn tilaan voidaan siirtyä vain yhdellä tietyllä syötemerkillä. Niinpä on oltava $x = A$, ja redusoinnin on täytynyt

perustua muotoa $A ::= ...Ax_1...x_n$ olevaan sääntöön, missä kuten edellä todettiin pätee $x_i \overset{*}{\Rightarrow} \varepsilon$, $1 \le i \le n$. Sääntö sisältää siis epäsuoran rekursion, kuten vaadittiin. Tämän jälkeen käsittelemällä tilasolmu *w*, luodaan funktion E-REDUSOIJA kautta tilasolmut samoilla tiloilla, kuin mitä solmut *u*…*v* sisältävät. □

Seuraava todistus perustuu Scottin ja muiden [SJH00] todistukseen siitä, että RNGLR-algoritmi pysähtyy kaikilla syötteillä.

Apulause 4 *Kaikille kontekstittomille kieliopeilla, jotka eivät sisällä epäsuoraa oikeaa rekursiota, muunneltu Tomitan algoritmi 2c pysähtyy kaikilla syötteillä.*

Todistus Oletetaan, että algorimissa käytetään LR(1)-jäsennystaulua, joka on muodostettu kieliopista Γ ja syötteestä $a_1...a_n$. Lasketaan ensin GSS:n koon yläraja.

Algoritmin askeleella *i* luoduissa tilasolmuissa sama tila voi esiintyä vain kerran (tämä on juuri se muutos, joka Tomitan algoritmiin 2b on tehty), joten jokaisella askeleella on korkeintaan *N* kappaletta tilasolmuja, missä *N* on DFA:n tilojen lukumäärä. Niinpä GSS:ssä on korkeintaan $(n + 1)N$ tilasolmua. Jokainen kaari solmusta $u \in U_{i,k}$ on ensimmäinen kaari kahden askeleen pituisen polun varrella johonkin solmuun $U_{j,l}$, missä $j \le i$ ja $l \le k$. SIIRTÄJÄ lisää korkeintaan yhden kaaren tilasolmusta ja kaikki muut kaaret lisäävät funktiot REDUSOIJA ja E-REDUSOIJA. REDUSOIJA tarkistaa kahden askeleen pituisen polun olemassaolon ennen kaaren lisäämistä. E-REDUSOIJA ei tee tätä eksplisiittisesti, mutta koska siinä luodaan kahden askeleen mittainen polku kahden tilasolmun välille johonkin tyhjään sääntöön perustuen ja jokainen sääntö esiintyy kieliopissa vain kerran, voi tietty kahden askeleen mittainen kahden solmun välillä esiintyä vain kerran. Niinpä GSS:ssä on korkeintaan yksi kahden askeleen mittainen polku joukosta $U_{i,k}$ jokaiseen $(i+1)N$ solmuun joukossa $U_{j,l}$, missä $j \le i$ ja $l \le k$. Jokaisesta tilasolmusta algoritmin askeleella *i* lähtee siten korkeintaan $(i+1)N$ kaarta, joten kaikista askeleen *i* aikana luoduista tilasolmuista lähtee GSS:ssä korkeintaan $(i+1)N^2$ kaarta. Askeleen *i* tilasolmuille on siis lapsinaan korkeintaan $(i+1)N^2$ symbolisolmua. Koko GSS:ssä on siten korkeintaan

$$(\sum_{i=0}^{n}(i+1)N^2) + (n+1)N = \frac{(n+1)((n+2)N^2 + 2N)}{2}$$

solmua (tila- ja symbolisolmua).

Jokaisella kaarella GSS:ssä on symbolisolmu joko lähtösolmuna tai tulosolmuna. GSS:n rakentamisprosessi takaa, että symbolisolmuilla on vain yksi edeltäjä, joten sellaisia kaaria joiden päätesolmu on symbolisolmu on yhtä paljon kuin on symbolisolmuja. GSS:n rakentamisprosessi takaa myös sen, että symbolisolmuista lähtevillä kaarilla on päätesolmut luotu samalla algoritmin askeleella i. Näitä voi olla siis vain N kappaletta. Koska GSS:ssä symbolisolmuja on korkeintaan $\sum_{i=0}^{n}(i+1)N^2$ kappaletta, koko GSS:ssä on siten korkeintaan

$$(N+1)\sum_{i=0}^{n}(i+1)N^2 = \frac{(n+1)(n+2)N^2(N+1)}{2}$$

kaarta.

Funktiossa TOIMIJA **jokaiselle**-silmukassa käsitellään yhtä tilasolmua joukosta A ja suoritetaan kaikki toiminnot toimintataulusta tähän tilaan liittyen. Toimintoja on äärellinen määrä, joten funktio TOIMIJA pysähtyy. REDUSOIJA poistaa yhden elementin joukosta R. REDUSOIJAN sisemmässä **jokaiselle**-silmukassa lisätään joukkoon R jokainen redusointi LR(1)-jäsennystaulun yhdestä kohdasta, ja näitä voi olla vain äärellinen määrä. Ulommassa **jokaiselle**-silmukassa iteroidaan kaikkien $2m-1$ pituisen polun päässä olevien tilasolmujen läpi. GSS voi sisältää syklejä, mutta tarkistus kahden askeleen pituisen polun olemassaolosta pitää huolen siitä, että suoritetaan äärellinen määrä toimintoja. Funktion E-REDUSOIJA **jokaiselle**-silmukassa käydään läpi joukko R_ε, johon ei lisätä silmukan sisällä alkioita, joten funktio E-REDUSOIJA pysähtyy. Funktion SIIRTÄJÄ **jokaiselle**-silmukassa käydään läpi alkiot joukosta Q, johon ei silmukan sisällä lisätä alkioita, joten tämäkin funktio pysähtyy. Niinpä sen osoittamiseksi, että algoritmi pysähtyy, on osoitettava, että **jokaiselle**-silmukka funktiossa JÄSENNÄ_SYMBOLI(i) pysähtyy jokaisella i:n arvolla. On siis osoitettava, että joukot A, R ja R_ε tyhjenevät. Funktiossa REDUSOIJA joukkoon A lisätään alkioita silloin, kun GSS:ään lisätään uusi tila- ja symbolisolmu ja kaksi kaarta ja joukkoon R silloin, kun lisätään symbolisolmu ja kaksi kaarta. Funktiossa E-REDUSOIJA joukkoon A lisätään uusi tila- ja symbolisolmu ja kaksi kaarta. Koska GSS:n koolla on yläraja, funktioiden REDUSOIJA ja E-REDUSOIJA on jossain vaiheessa lopettava lisäämästä alkioita joukkoihin A, R ja R_ε. Lopulta siis $A = R = R_\varepsilon = \theta$ ja algoritmi pysähtyy. □

Apulauseen 4 mukaan muunneltu Tomitan algoritmi 2c pysähtyy siis kieliopeilla, jotka eivät sisällä epäsuoraa oikeaa rekursiota. Alkuperäisessä

Tomitan algoritmissa pysähtymättömyysongelmat liittyivät kielioppeihin, jotka sisältävät epäsuoran vasemmanpuoleisen rekursion. Edellä on huomattu, että epäsuoran oikean rekursion sisältävän kieliopin kanssa muunneltu Tomitan algoritmi 2c toimii kuten alkuperäinen Tomitan algoritmi 2, joten näiden seikkojen perusteella esitetään seuraava teoreema.

Teoreema 3 *Muunneltu Tomitan algoritmi 2c pysähtyy kaikilla syötteillä.*

11.2. Muunneltu Tomitan algoritmi 2c on rajoittamaton suuruusluokaltaan

Osoitetaan seuraavaksi, että muunnellun Tomitan algoritmin 2 aikavaatimus on suuruusluokaltaan vähintään $O(n^4)$. Todistus perustuu Scottin ja muiden [SJE04] todistukseen siitä, että RNGLR-jäsentäjä on vähintään $O(n^4)$. Todistuksessa käytettävä kielioppi ei sisällä tyhjiä sääntöjä, joten tulos on sama alkuperäiselläkin Tomitan algoritmilla 2. Kielioppi kuuluu joukkoon Johnsonin [Joh91] esittämiä kielioppeja, joilla Johnson osoitti alkuperäisen Tomitan algoritmin olevan rajoittamatonta polynomista suuruusluokkaa. Johnsonin perustelut perustuivat kuitenkin algoritmin tulostukseen. Seuraavaksi osoitetaan, että algoritmi itsessään on suuruusluokaltaan kuutiollista suurempi.

Teoreema 4 *Muunneltu Tomitan algoritmi 2c on suuruusluokaltaan vähintään* $O(n^4)$.

Todistus Tarkastellaan kielioppia Γ_1, jonka säännöt ovat

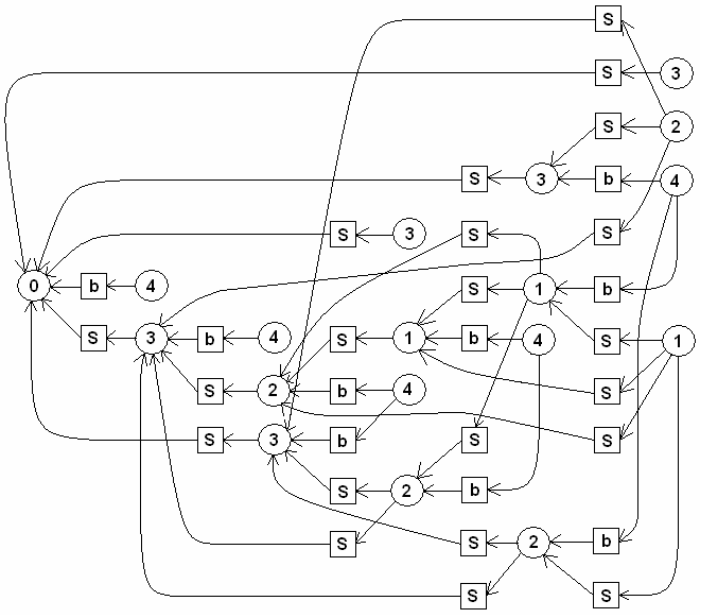
$S ::= SS$
$S ::= SSS$
$S ::= b.$

Tästä muodostettu LR(1)-jäsennystaulu on esitetty kuvassa 11.5.

	$	b	S
0		s4	g3
1	r1/r2	r1/r2/s4	g1
2	r1	r1/s4	g1
3	hyv.	s4	g2
4	r3	r3	

Kuva 11.5. LR(1)-jäsennystaulu kieliopille Γ_1.

Kuvassa 11.6 on esitetty syötteestä *bbbbb* saatu GSS



Kuva 11.6. Täydellinen GSS syötteelle *bbbbb*.

GSS:stä huomataan, että jokaisesta tilasolmusta tilalla 1 on kahden askeleen mittainen polku jokaiseen tilasolmuun tilalla 1 alemmilla tasoilla. Käytetään tätä ominaisuutta osoittamaan, että muunneltu Tomitan algoritmi 2 on aikakompleksisuudeltaan luokkaa $O(n^4)$, kun kieliopista muodostetulle jäsentäjälle Γ annetaan syötteeksi b^n.

Funktio REDUSOIJA suoritetaan jokaiselle alkiolle (u, p) joukossa R. Ylin **jokaiselle**-silmukka etsii tällöin kaikki tilasolmut, jotka ovat 2p-1 pituisen polun päässä solmusta u. Näitä on korkeintaan $O(i)$ kappaletta. Osoitetaan seuraavaksi, että kieliopille Γ, syvyys-ensin-etsintä kaikkien solmujen löytämiseksi on aikakompleksisuudeltaan $O(i^3)$.

Ensin osoitetaan, että $U_{i,0}$ sisältää tilasolmun w_i, kun i > 3, ja että jokaisesta solmusta w_i on kahden askeleen mittainen polku solmuun w_j, $3 \le j < i$.

Aluksi luodaan tilasolmu [v_0, 0] joukkoon $U_{0,0}$. Joukko $U_{0,0}$ asetetaan joukkoon A ja funktiossa TOIMIJA alkio (v_0, 4) joukkoon Q. Funktio SIIRTÄJÄ luo tilasolmun [v_1, 4], joka asetetaan joukkoon $U_{1,0}$. TOIMIJA asettaa joukkoon R alkion (u_0, 1) , missä u_0 on solmujen v_0 ja v_1 välissä oleva symbolisolmu. REDUSOIJA poistaa alkion joukosta R ja luo solmun v_2 tilalla 3, joka asetetaan joukkoon $U_{1,0}$. TOIMIJA asettaa alkion (v_2, 4) joukkoon Q ja SIIRTÄJÄ luo

tilasolmun [v_3, 4], joka asetetaan joukkoon $U_{2,0}$. TOIMIJA asettaa alkion (u_1, 1) joukkoon *R*, missä u_1 on solmujen v_2 ja v_3 välissä oleva symbolisolmu. REDUSOIJA poistaa alkion joukosta *R* ja luo tilasolmun [v_4, 2]. TOIMIJA asettaa alkion (v_4, 4) joukkoon *Q*, josta SIIRTÄJÄ poistaa alkion, ja luo tilasolmun [v_6, 4], joka asetetaan joukkoon $U_{3,0}$. TOIMIJA asettaa alkion (u_6, 1) joukkoon *R*. Kun REDUSOIJA poistaa tämän alkion joukosta *R*, luodaan tilasolmu [v_7, 1], ja tämä solmu asetetaan joukkoon $U_{3,0}$, kuten vaadittiin.

Oletetaan, että joukossa on $U_{j,0}$ on tilasolmu [w_j, 1], $3 \le j < i$, ja että solmusta w_j on kahden askeleen mittainen polku tilasolmuun [w_q, 1], $3 \le q < j$. Kun luodaan tilasolmu [w_{i-1}, 1] joukkoon $U_{i-1,0}$, (w_{i-1}, 4) lisätään joukkoon *Q* seuraavan kerran kun funktio TOIMIJA suoritetaan. Kun suoritetaan SIIRTÄJÄ(*i*-1), luodaan tilasolmu [v, 4] joukkoon $U_{i,0}$ ja symbolisolmu *u*. TOIMIJA lisää tämän jälkeen alkion (*u*, 3) joukkoon *R*. Kun tämän jälkeen suoritetaan funktio REDUSOIJA, lisätään tilasolmu [w_i, 1] joukkoon $U_{i,0}$ ja luodaan kahden askeleen mittainen polku solmusta w_i solmuun w_{i-1}, jos tälläistä ei vielä siellä ole. Suoritettaessa TOIMIJA seuraavan kerran, lisätään joukkoon *R* alkio (*t*, 1), missä [*t*, *S*] on symbolisolmu. Koska jokaisesta solmusta w_{i-1} tilalla 1 on kahden askeleen mittaiset polut jokaiseen solmuun w_j, $3 \le j < i-1$ ja $g1$ on LR(1)-jäsennystaulun kohdassa (1, *S*), käsiteltäessä alkio (*t*, 1) joukosta *R*, luodaan kahden askeleen mittaiset polut jokaisesta solmusta w_i jokaiseen solmuun w_j, niinkuin vaadittiin.

Algoritmin askeleella *i* luotaessa kahden askeleen mittainen polku solmusta w_i solmuun w_q, $3 \le q < i-1$, joukkoon *R* lisätään alkio (*s*, 2), missä *s* on polun keskellä oleva symbolisolmu symbolilla *S*. Kun (*s*, 2) käsitellään, käytettäessä tavallista syvyys-ensin -etsintää etsittäessä solmut, jotka ovat 5 askeleen päässä solmusta w_q, on kuljettava 2(*q* – 3) + 1 kaaren kautta solmusta w_q jokaiseen solmuun w_l, $3 \le l < q-1$ ja solmusta w_l eteenpäin, kunnes 5 askeleen mittainen polku on kuljettu. Näin ollen kuljetaan ainakin

$$2(q-3)+1+\sum_{3}^{q-1}(l-3)=\frac{(q-2)(q-3)}{2}+(q-2)$$

kaaren kautta funktiossa REDUSOIJA joukon *R* alkiota (*s*, 2) kohti. Algoritmin askeleella *i* funktio REDUSOIJA suoritetaan jokaiselle *i* – 2 alkiolle (*s*, 2), joten kuljettujen kaarien lukumäärä tällä algoritmin askeleella on vähintään

$$\sum_{3}^{i-1}\frac{(q-2)(q-3)}{2}+(q-2)=\frac{i^3-6i^2+11i-6}{6}$$

Algoritmissa suoritetaan *n* askelta, joten

$$\sum_{0}^{n} i^3 = \frac{n^2(n+1)^2}{4},$$

eli Tomitan algoritmin aikakompleksisuus on vähintään $O(n^4)$. □

Teoreeman 4 tavalla tarkastelemalla kielioppia, jonka säännöt ovat

$S' ::= S$
$S ::= b \mid S^{N-1},$

huomataan algoritmin olevan suuruusluokaltaan vähintään luokkaa $O(d^N)$. Tämä huomio johtaa teoreemaan 5.

Teoreema 5 *Muunneltu Tomitan algoritmi 2c on aikavaativuudeltaan rajoittamaton.*

12. Yhteenveto

Tässä tutkielmassa tarkasteltiin yleistettyä jäsennystekniikkaa, joka perustuu perinteiseen LR-jäsentämisalgoritmiin.

Työ aluksi esiteltiin yleistetyn LR-jäsentämisen kehitys 1960-luvun alkupuolelta nykypäivään. Todettiin, että Tomita kehitti käytännön sovelluksen sille viitekehykselle, jonka Lang oli aikaisemmin kuvannut. Tomitan päasiallisena tuloksena oli kehittää tietorakenne, graafirakenteinen pino, GSS, johon tallennetaan kaikki pinot, jotka LR-jäsennysalgoritmi mahdollistaa.

Työssä esiteltiin GLR-jäsentäjän toimintaa ensin yleisellä tasolla, sitten annettiin formaali kuvaus 1e-algoritmin muodossa. 1e-algoritmin todettiin hylkäävän joitakin lauseita, jotka kuuluvat kieliopin määräämään kieleen. Tämän todettiin johtuvan sellaisista kieliopeista, joiden säännöt sisältävät epäsuoran oikeanpuoleisen rekursion.

Työssä esitettiin Tomitan ratkaisu ongelmaan, mutta tämän ratkaisun huomattiin vievän algoritmin päättymättömään silmukkaan, kun se käsittelee kielioppia, joka sisältää epäsuoran vasemmanpuoleisen rekursion.

Työssä esiteltiin tämän jälkeen kaksi ratkaisua, joiden todettiin toimivan oikein kieliopeilla, jotka sisältävät epäsuoran oikeanpuoleisen rekursion.

Farshin ratkaisu tilanteeseen oli suhteellisen ilmeinen korjaus, joka toi mukanaan epämiellyttäviä rasitteita algoritmin suorituskyvyn kannalta. Scottin ja muiden ratkaisu sisälsi hienovaraisemman korjauksen Tomitan algoritmiin, jossa LR(1)-jäsennystaulua muokattiin redusointien osalta.

Tämän jälkeen tutkielmassa analysoitiin tarkemmin alkuperäisessä Tomitan algoritmissa olleita ongelmia ja esitettiin kirjoittajan omia korjauksia.

Tomita rakensi algoritminsa vaiheittain. Samoin tutkielmassa esitetyt korjaukset Tomitan algoritmiin tehtiin vaiheittain.

Ensin algoritmia korjattiin niin, että se pystyy käsittelemään kielioppeja, jotka sisältävät epäsuoran vasemmanpuoleisen rekursion joutumatta päättymättömään silmukkaan. Seuraavaksi algoritmia korjattiin niin, että se pystyy käsittelemään syklisiä kielioppeja. Lopuksi algoritmia muokattiin niin, että se toimii jälleen epäsuoran oikean rekursion sisältävillä kieliopeilla.

Työn lopuksi todistettiin muutamia tuloksia muunneltuun Tomitan algoritmiin liittyen.

Viiteluettelo

[AJU75] Alfred V. Aho, S. C. Johnson, and Jeffrey D. Ullman. Deterministic parsing of ambiguous grammars. *Communications of the ACM* **18**, 8 (August 1975) 441—452.

[ASU86] Alfred V. Aho, Ravi Sethi and Jeffrey D. Ullman. *Compilers: Principles Techniques and Tools*. Addison-Wesley, 1986.

[AJH01] J. Aycock, R.N. Horspool, J. Janousek and B. Melichar. Even faster generalized LR parsing. *Acta Informatica* **37** (2001) 633-631.

[BBG+63] J. W. Backus, F. L. Bauer, J. Green, C. Katz, J. McCarthy, A. J. Perlis, H. Rutishauser, K. Samelson, B. Vauqois, J. HWegstein, A. van Wijngaarden, and M. Woodger. Revised report on the programming language ALGOL 60. *Communications of the ACM* **6,** 1 (January 1963) 1–24.

[DeR69] Franklin L. DeRemer. *Practical Translators for LR(k) Languages*. PhD thesis, M.I.T., 1969.

[DeR71] Franklin L. DeRemer. Simple LR(*k*) grammars. *Communications of the ACM* **14**, 7 (1971) 453-460.

[GR62] Seymour Ginsburg and H. Gordon Rice. Two families of languages related to ALGOL. *Journal of the ACM* **9**, 3 (1962) 350–371.

[Joh75] S. C. Johnson. YACC — yet another compiler-compiler. Technical Report 32, AT&T Bell Laboratories, 1975.

[Joh91] Mark Johnson. The computational complexity of GLR parsing. In: Masaru Tomita (ed), G*eneralized LR parsing*, pages 35-42. Kluwer Academic Publishers, 1991.

[JS02] Adrian Johnstone and Elizabeth Scott. Generalised reduction modified LR parsing for domain specific language prototyping. In *Proc. 35th Annual Hawaii International Conference On System Sciences (HICSS02)*. IEEE Computer Society, IEEE, 2002.

[JSE04] Adrian Johnstone, Elizabeth Scott and Giorgios Economopoulos. Generalised Parsing: Some Costs. In: Evelyn Duesterwald (ed), *Compiler Construction, 13th International Conference, CC 2004*, pages 89-103. Springer, 2004.

[Knu65] Donald E. Knuth. On the translation of languages from left to right. *Information and Control* **8**, 6 (1965) 607-639) 1965.

[Lan74] Bernard Lang. Deterministic techniques for efficient non-deterministic parsers. In *Automata, Languages and Programming: 2nd Colloquim*, pages 255-269. Springer-Verlag, 1974.

[MN04] Scott McPeak and George C. Necula. Elkhound: A Fast, practical GLR parser generator. In: Evelyn Duesterwald (ed), *Compiler Construction, 13th International Conference, CC 2004*, pages 73-88. Springer, 2004.

[NF91] Rahman Nozonoor-Farshi. GLR parsing for ε-grammars. In: Masaru Tomita (ed), G*eneralized LR parsing*, pages 60-75. Kluwer Academic Publishers, 1991.

[Rek92] Jan G. Rekers. Parser generation for interactive environments. PhD Thesis, University of Amsterdam, 1992.

[SJH00] Elizabeth Scott, Adrian Johstone and Shamsa Sadaf Hussain. Tomita-style generalised LR parsers. Technical Report TR-00-12, Royal Holloway, University of London, Computer Science Department, December 2000.

[SJE03] Elizabeth Scott, Adrian Johstone and Giorgios Economopoulos. BRN-table based GLR parsers. Technical Report TR-03-06, Royal Holloway, University of London, Computer Science Department, June 2003.

[Tom86] Masaru Tomita. *Efficient Parsing for Natural Language*. Kluwer Academic Publishers, 1986.

[vdBHdJ+ar] Mark van den Brand, Jan Heering, Haycode Jong, Merijn de Jonge, Tobias Kuipers, Paul Klint, Leon Moonen, Pieter Olivier,

Jeroen Scheerder, Jurgen Vinju, Eelco Visser, and JoostVisser. The ASF+SDF meta-environment: a component-based language development environment. In *Compiler Construction: 10thInternational Conference, CC 2001*. Springer-Verlag, 2001.

[Vis97] Eelco Visser. Syntax definition for language prototyping. PhD Thesis, University of Amsterdam, 1997.